



河南大學

Henan University

I'm dying to play GTA 6

Contestant

张帆

Fan Zhang

范恒嘉

Hengjia Fan

宋明贤

Mingxian Song

目录

1	数据结构	1
1.1	Chtholly	1
1.2	DSU	2
1.3	Fenwick	2
1.4	SegmentTree(动态开点)	4
1.5	SegmentTree	7
1.6	SparseTable	9
1.7	treap	10
2	图论	11
2.1	HLD	11
2.2	HLD extend	13
2.3	HopcroftKarp	18
2.4	LCA	19
2.5	TarjanEBCC	21
2.6	TarjanSCC	22
3	字符串	24
3.1	01tire	24
3.2	AhoCorasick	25
3.3	PAM	27
3.4	StringHash	28
3.5	Z	30
3.6	manacher	30
3.7	trie	31
4	数学	32
4.1	BigNum	32
4.2	Comb	41
4.3	FFT	42
4.4	FWT	44
4.5	LinearBasis	45
4.6	LinearSieve	47
4.7	MTT	48

4.8	Matrix	53
4.9	NTT	54
4.10	fast pow	58
4.11	sqrt mod	58
5	附录	60
5.1	VSCode 设置	60
5.2	互质的规律	60
5.3	常数表	60
5.4	常见错因	61
5.5	斐波那契数列	61
5.6	算法	61
5.7	组合数学公式	63
5.8	编译参数	64
5.9	运行脚本	64
5.10	随机素数	64

1 数据结构

1.1 Chtholly

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct Chtholly
20 {
21     map<int, pair<int, int>> internals;
22     unordered_map<int, int> cnt;
23
24     Chtholly() {}
25
26     void add(int l, int r, int val)
27     {
28         internals[l] = {r, val};
29         cnt[val] = r - l + 1;
30     }
31
32     void split(int pos)
33     {
34         auto it = internals.lower_bound(pos);
35
36         if (it == internals.begin())
37             return ;
38
39         it--;
40
41         int L = it->first;
42         auto [R, val] = it->second;
```

```
44         if (L == pos)
45             return ;
46
47         if (R < pos)
48             return ;
49
50         internals[L] = {pos - 1, val};
51         internals[pos] = {R, val};
52     }
53
54     void assign(int l, int r, int val)
55     {
56         split(l);
57         split(r + 1);
58
59         unordered_set<int> toerase;
60         for (auto it = internals.lower_bound(l); it != internals.end(); it++)
61         {
62             int L = it->first;
63             auto [R, x] = it->second;
64
65             if (r < L)
66                 break;
67
68             toerase.insert(L);
69             cnt[x] -= (R - L + 1);
70         }
71
72         for (auto temp : toerase)
73             internals.erase(temp);
74
75         cnt[val] += (r - l + 1);
76         internals[l] = {r, val};
77     }
78 };
79 // snippet-end
80
81 void solve()
82 {
83
84 }
85
86 signed main()
87 {
88     // ios::sync_with_stdio(false);
89     // cout.tie(nullptr);
90     // cin.tie(nullptr);
91     int T = 1;
```

```

92 // cin >> T;
93 while (T--)
94     solve();
95 return 0;
96 }

```

1.2 DSU

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct DSU
20 {
21     vector<int> f, siz;
22
23     DSU(int n) : f(n), siz(n, 1)
24     {
25         iota(f.begin(), f.end(), 0);
26     }
27
28     int find(int x)
29     {
30         while (f[x] != x)
31             x = f[x] = f[f[x]];
32         return x;
33     }
34
35     bool merge(int x, int y)
36     {
37         x = find(x);
38         y = find(y);
39

```

```

40         if (x == y)
41             return false;
42
43         if (siz[x] < siz[y])
44             swap(x, y);
45
46         siz[x] += siz[y];
47         f[y] = x;
48         return true;
49     }
50
51     int size(int x)
52     {
53         return siz[find(x)];
54     }
55
56     bool connected(int x, int y)
57     {
58         return find(x) == find(y);
59     }
60 };
61 // snippet-end
62
63 void solve()
64 {
65 }
66
67 signed main()
68 {
69     // ios::sync_with_stdio(false);
70     // cout.tie(nullptr);
71     // cin.tie(nullptr);
72     int T = 1;
73     // cin >> T;
74     while (T--)
75         solve();
76     return 0;
77 }
78 }

```

1.3 Fenwick

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;

```

```

7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin: Fenwick, 区间修改 + 区间查询
19 struct Fenwick
20 {
21     int n;
22     vector<int> a, b;
23
24     Fenwick(int _n) : n(_n), a(_n + 1), b(_n + 1) {}
25
26     int lowbit(int x)
27     {
28         return x & -x;
29     }
30
31     void modify(int x, int k, vector<int> &v)
32     {
33         while (x >= 1 && x <= n)
34         {
35             v[x] += k;
36             x += lowbit(x);
37         }
38     }
39
40     // (r + 1) * (a[1] + ... + a[r]) - (b[1] * 1 + ... + b[r] * r)
41     void update(int l, int r, int k)
42     {
43         modify(l, k, a);
44         modify(r + 1, -k, a);
45
46         modify(l, k * l, b);
47         modify(r + 1, -k * (r + 1), b);
48     }
49
50     int get(int x, vector<int> &v)
51     {
52         int res = 0;
53         while (x > 0)

```

```

54     {
55         res += v[x];
56         x -= lowbit(x);
57     }
58
59     return res;
60 }
61
62 int query(int l, int r)
63 {
64     if (l > r)
65         return 0ll;
66
67     int R = (r + 1) * get(r, a) - get(r, b);
68     int L = l * get(l - 1, a) - get(l - 1, b);
69
70     return R - L;
71 }
72 };
73 // snippet-end
74
75 // snippet-begin: Fenwick, 单点修改 + 区间查询 + 第k小值
76 struct Fenwick
77 {
78     int n;
79     vector<int> a;
80
81     Fenwick(int _n) : n(_n), a(_n + 1) {}
82
83     int lowbit(int x)
84     {
85         return x & -x;
86     }
87
88     void update(int x, int k)
89     {
90         while (x >= 1 && x <= n)
91         {
92             a[x] += k;
93             x += lowbit(x);
94         }
95     }
96
97     int pre(int r)
98     {
99         int res = 0;
100
101         while (r > 0)

```

```

102     {
103         res += a[r];
104         r -= lowbit(r);
105     }
106
107     return res;
108 }
109
110 int query(int l, int r)
111 {
112     return pre(r) - pre(l - 1);
113 }
114
115 int kth(int k)
116 {
117     int ans = 0;
118     for (int p = __lg(n); p >= 0; p--)
119     {
120         int step = 1ll << p;
121         if (ans + step <= n && a[ans + step] < k)
122         {
123             k -= a[ans + step];
124             ans += step;
125         }
126     }
127
128     return ans + 1;
129 }
130 };
131 // snippet-end
132
133 void solve()
134 {
135 }
136 }
137
138 signed main()
139 {
140     // ios::sync_with_stdio(false);
141     // cout.tie(nullptr);
142     // cin.tie(nullptr);
143     int T = 1;
144     // cin >> T;
145     while (T--)
146         solve();
147     return 0;
148 }

```

1.4 SegmentTree(动态开点)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = int64_t;
4 using u64 = uint64_t;
5 using f64 = long double;
6 using i128 = __int128_t;
7 using u128 = __uint128_t;
8
9 const long double eps = 1e-12;
10 const i64 mod = 1e9 + 7;
11 const i64 INF = 1e18;
12 const int inf = 1e9;
13
14 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
15 auto rnd = [](i64 l, i64 r) { return (l <= r ? uniform_int_distribution<i64>(l, r)(
    rng) : 0); };
16
17 // snippet-begin: SegmentTree, 动态开点
18 #define ls (node->l)
19 #define rs (node->r)
20
21 template<typename Info, typename Tag>
22 struct SegmentTree
23 {
24     struct Node
25     {
26         Info info = Info(0);
27         Tag lazy = Tag();
28
29         Node *l = nullptr;
30         Node *r = nullptr;
31     };
32
33     int n;
34     Node *root;
35     SegmentTree(int _n) : n(_n), root(nullptr) {}
36
37     void newNode(Node *&node, int start, int end)
38     {
39         if (node) return;
40         node = new Node();
41         node->info.len = end - start + 1;
42     }
43
44     void pushdown(Node *node, int start, int end)
45     {

```

```

46     if (node->lazy.empty() || start == end) return;
47
48     int mid = (start + end) / 2;
49     newNode(ls, start, mid);
50     newNode(rs, mid + 1, end);
51
52     node->lazy.apply(ls->info);
53     ls->lazy.merge(node->lazy);
54
55     node->lazy.apply(rs->info);
56     rs->lazy.merge(node->lazy);
57
58     node->lazy = Tag();
59 }
60
61 void pushup(Node *&node, int start, int end)
62 {
63     Info l = (ls ? ls->info : Info());
64     Info r = (rs ? rs->info : Info());
65     node->info = l + r;
66     node->info.len = end - start + 1;
67 }
68
69 void update(Node *&node, int start, int end, int l, int r, const Tag &val)
70 {
71     if (r < start || l > end) return;
72     newNode(node, start, end);
73
74     if (l <= start && end <= r)
75     {
76         val.apply(node->info);
77         node->lazy.merge(val);
78         return;
79     }
80
81     pushdown(node, start, end);
82     int mid = (start + end) / 2;
83     if (l <= mid) update(ls, start, mid, l, r, val);
84     if (mid < r) update(rs, mid + 1, end, l, r, val);
85     pushup(node, start, end);
86 }
87
88 void modify(Node *&node, int start, int end, int pos, const Info &val)
89 {
90     if (pos < start || pos > end) return;
91     newNode(node, start, end);
92
93     if (start == end)

```

```

94     {
95         node->info = val;
96         node->info.len = 1;
97         node->lazy = Tag();
98         return;
99     }
100
101     pushdown(node, start, end);
102     int mid = (start + end) / 2;
103     if (pos <= mid)
104         modify(ls, start, mid, pos, val);
105     else
106         modify(rs, mid + 1, end, pos, val);
107     pushup(node, start, end);
108 }
109
110 Info query(Node *&node, int start, int end, int l, int r)
111 {
112     if (!node || r < start || l > end) return Info();
113     if (l <= start && end <= r) return node->info;
114
115     pushdown(node, start, end);
116     int mid = (start + end) / 2;
117     return query(ls, start, mid, l, r) + query(rs, mid + 1, end, l, r);
118 }
119
120 void update(int l, int r, const Tag &val)
121 {
122     if (l > r) return;
123     update(root, 0, n - 1, l, r, val);
124 }
125
126 void modify(int pos, const Info &val)
127 {
128     modify(root, 0, n - 1, pos, val);
129 }
130
131 Info query(int l, int r)
132 {
133     if (l > r) return Info();
134     return query(root, 0, n - 1, l, r);
135 }
136 };
137
138 struct info
139 {
140     i64 mx = -INF;
141     i64 mn = INF;

```

```

142     i64 sum = 0;
143     i64 ssum = 0;
144     int len = 0;
145
146     info () : mx(-INF), mn(INF), sum(0), ssum(0), len(0) {};
147     info (i64 val) : mx(val), mn(val), sum(val), ssum(val * val), len(1) {};
148 };
149
150 info operator+(const info &l, const info &r)
151 {
152     info res;
153     res.mx = max(l.mx, r.mx);
154     res.mn = min(l.mn, r.mn);
155     res.sum = l.sum + r.sum;
156     res.ssum = l.ssum + r.ssum;
157     res.len = l.len + r.len;
158
159     return res;
160 }
161
162 // // 区间加
163 // struct tagAdd
164 // {
165 //     i64 add = 0;
166
167 //     tagAdd() : add(0) {}
168 //     tagAdd(i64 _add) : add(_add) {}
169
170 //     bool empty() const
171 //     {
172 //         return add == 0;
173 //     }
174
175 //     void apply(info &a) const
176 //     {
177 //         i64 old = a.sum;
178
179 //         a.mx += add;
180 //         a.mn += add;
181 //         a.sum += add * a.len;
182 //         a.ssum += 2 * add * old + add * add * a.len;
183 //     }
184
185 //     void merge(const tagAdd &o)
186 //     {
187 //         if (o.empty())
188 //             return;
189

```

```

190 //         add += o.add;
191 //     }
192 // };
193
194 // // 区间赋值
195 // struct tagAssign
196 // {
197 //     bool has = false;
198 //     i64 val = 0;
199
200 //     tagAssign() : has(false), val(0) {};
201 //     tagAssign(i64 _val) : has(true), val(_val) {};
202
203 //     bool empty() const
204 //     {
205 //         return !has;
206 //     }
207
208 //     void apply(info &a) const
209 //     {
210 //         a.mx = val;
211 //         a.mn = val;
212 //         a.sum = val * a.len;
213 //         a.ssum = val * val * a.len;
214 //     }
215
216 //     void merge(const tagAssign &o)
217 //     {
218 //         if (!o.has)
219 //             return;
220
221 //         has = true;
222 //         val = o.val;
223 //     }
224 // };
225
226 #undef ls
227 #undef rs
228 // snippet-end:
229
230 void solve()
231 {
232
233 }
234
235 int main()
236 {
237     // ios::sync_with_stdio(false);

```



```

238 // cout.tie(nullptr);
239 // cin.tie(nullptr);
240 int T = 1;
241 // cin >> T;
242 while (T--)
243     solve();
244 return 0;
245 }

```

1.5 SegmentTree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 #define ls (node * 2 + 1)
20 #define rs (node * 2 + 2)
21
22 template<typename Info, typename Tag>
23 struct SegmentTree
24 {
25     int n;
26     vector<Info> tree;
27     vector<Tag> lazy;
28
29     SegmentTree() {}
30
31     SegmentTree(int _n) : n(_n)
32     {
33         init(vector<Info>(n, Info(0)));
34     }
35
36     template<typename T>

```

```

37 SegmentTree(const vector<T> &input) : n(input.size())
38 {
39     init(input);
40 }
41
42 template<typename T>
43 void init(const vector<T> &input)
44 {
45     n = input.size();
46     tree.resize(4 * n + 5, Info());
47     lazy.resize(4 * n + 5, Tag());
48     build(0, 0, n - 1, input);
49 }
50
51 template<typename T>
52 void build(int node, int start, int end, const vector<T> &input)
53 {
54     if (start == end)
55     {
56         tree[node] = input[start];
57     }
58     else
59     {
60         int mid = (start + end) / 2;
61         build(ls, start, mid, input);
62         build(rs, mid + 1, end, input);
63         tree[node] = tree[ls] + tree[rs];
64     }
65 }
66
67 void pushdown(int node)
68 {
69     if (lazy[node].empty()) return;
70
71     lazy[node].apply(tree[ls]);
72     lazy[node].apply(tree[rs]);
73
74     lazy[ls].merge(lazy[node]);
75     lazy[rs].merge(lazy[node]);
76
77     lazy[node] = Tag();
78 }
79
80 void update(int node, int start, int end, int l, int r, const Tag &val)
81 {
82     if (end < l || start > r) return;
83     if (l <= start && end <= r)
84     {

```

```

85     val.apply(tree[node]);
86     lazy[node].merge(val);
87     return;
88 }
89
90 pushdown(node);
91 int mid = (start + end) / 2;
92 if (l <= mid) update(ls, start, mid, l, r, val);
93 if (mid < r) update(rs, mid + 1, end, l, r, val);
94 tree[node] = tree[ls] + tree[rs];
95 }
96
97 void modify(int node, int start, int end, int pos, const Info &val)
98 {
99     if (start == end)
100     {
101         tree[node] = val;
102         return;
103     }
104
105     pushdown(node);
106     int mid = (start + end) / 2;
107     if (pos <= mid) modify(ls, start, mid, pos, val);
108     else if (pos > mid) modify(rs, mid + 1, end, pos, val);
109     tree[node] = tree[ls] + tree[rs];
110 }
111
112 Info query(int node, int start, int end, int l, int r)
113 {
114     if (l > end || r < start) return Info();
115     if (l <= start && end <= r) return tree[node];
116     pushdown(node);
117     int mid = (start + end) / 2;
118     return query(ls, start, mid, l, r) + query(rs, mid + 1, end, l, r);
119 }
120
121 void update(int l, int r, const Tag &val)
122 {
123     if (l > r) return;
124     update(0, 0, n - 1, l, r, val);
125 }
126
127 void modify(int pos, const Info &val)
128 {
129     modify(0, 0, n - 1, pos, val);
130 }
131
132 Info query(int l, int r)

```

```

133 {
134     if (l > r) return Info();
135     return query(0, 0, n - 1, l, r);
136 }
137 };
138
139 struct info
140 {
141     i64 mx = -INF;
142     i64 mn = INF;
143     i64 sum = 0;
144     i64 ssum = 0;
145     int len = 0;
146
147     info () : mx(-INF), mn(INF), sum(0), ssum(0), len(0) {};
148     info (i64 val) : mx(val), mn(val), sum(val), ssum(val * val), len(1) {};
149 };
150
151 info operator+(const info &l, const info &r)
152 {
153     info res;
154     res.mx = max(l.mx, r.mx);
155     res.mn = min(l.mn, r.mn);
156     res.sum = l.sum + r.sum;
157     res.ssum = l.ssum + r.ssum;
158     res.len = l.len + r.len;
159
160     return res;
161 }
162
163 // // 区间加
164 // struct tagAdd
165 // {
166 //     i64 add = 0;
167
168 //     tagAdd() : add(0) {}
169 //     tagAdd(i64 _add) : add(_add) {}
170
171 //     bool empty() const
172 //     {
173 //         return add == 0;
174 //     }
175
176 //     void apply(info &a) const
177 //     {
178 //         i64 old = a.sum;
179
180 //         a.mx += add;

```

```

181 //      a.mn += add;
182 //      a.sum += add * a.len;
183 //      a.ssum += 2 * add * old + add * add * a.len;
184 //  }
185
186 //  void merge(const tagAdd &o)
187 //  {
188 //      if (o.empty())
189 //          return;
190
191 //      add += o.add;
192 //  }
193 // };
194
195 // // 区间赋值
196 // struct tagAssign
197 // {
198 //     bool has = false;
199 //     i64 val = 0;
200
201 //     tagAssign() : has(false), val(0) {};
202 //     tagAssign(i64 _val) : has(true), val(_val) {};
203
204 //     bool empty() const
205 //     {
206 //         return !has;
207 //     }
208
209 //     void apply(info &a) const
210 //     {
211 //         a.mx = val;
212 //         a.mn = val;
213 //         a.sum = val * a.len;
214 //         a.ssum = val * val * a.len;
215 //     }
216
217 //     void merge(const tagAssign &o)
218 //     {
219 //         if (!o.has)
220 //             return;
221
222 //         has = true;
223 //         val = o.val;
224 //     }
225 // };
226
227 #undef ls
228 #undef rs

```

```

229 // snippet-end
230
231 void solve()
232 {
233
234 }
235
236 signed main()
237 {
238     // ios::sync_with_stdio(false);
239     // cout.tie(nullptr);
240     // cin.tie(nullptr);
241     int T = 1;
242     // cin >> T;
243     while (T--)
244         solve();
245     return 0;
246 }

```

1.6 SparseTable

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 template<typename T>
20 struct SparseTable
21 {
22     T op(T a, T b) { return max(a, b); }
23     vector<vector<T>>> a;
24     int n;
25
26     SparseTable(vector<T>& input)

```

```

27 {
28     n = input.size();
29
30     int max_log = __lg(n);
31     a.assign(n, vector<T>(max_log + 1, 0));
32
33     for (int i = 0; i < n; i++)
34         a[i][0] = input[i];
35
36     for (int j = 1; j <= max_log; j++)
37     {
38         for (int i = 0; i + (1 << j) - 1 < n; i++)
39         {
40             // [i, i + 2 ^ (j - 1) - 1], [i + 2 ^ (j - 1), i + 2 ^ j - 1];
41             a[i][j] = op(a[i][j - 1], a[i + (1 << (j - 1))][j - 1]);
42         }
43     }
44 }
45
46 T query(int l, int r)
47 {
48     assert(l <= r && l >= 0 && r < n);
49
50     int j = __lg(r - l + 1);
51     // [l, l + 2 ^ j - 1], [r - 2 ^ j + 1, r];
52     return op(a[l][j], a[r - (1 << j) + 1][j]);
53 }
54 };
55 // snippet-end
56
57 void solve()
58 {
59     int n, m;
60     cin >> n >> m;
61     vector<int> a(n + 1);
62     for (int i = 1; i <= n; i++)
63         cin >> a[i];
64
65     SparseTable st(a);
66     while (m--)
67     {
68         int l, r;
69         cin >> l >> r;
70         cout << st.query(l, r) << "\n";
71     }
72 }
73
74 signed main()

```

```

75 {
76     ios::sync_with_stdio(false);
77     cout.tie(nullptr);
78     cin.tie(nullptr);
79     int T = 1;
80     // cin >> T;
81     while (T--)
82         solve();
83     return 0;
84 }

```

1.7 treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
17     rng) : 0); };
18
19 // snippet-begin:
20 struct Node
21 {
22     Node *left = nullptr, *right = nullptr;
23     pair<int, int> key;
24     i64 priority;
25     int min_left = INT32_MAX;
26     Node(pair<int, int> key) : key(key), priority(rng()), min_left(key.second) {}
27 };
28
29 int get_left_min(Node *cur)
30 {
31     return cur == nullptr ? INT32_MAX : cur->min_left;
32 }
33
34 void update(Node *cur)

```

```

35     if (cur != nullptr)
36         cur->min_left = min(cur->key.second, min(get_left_min(cur->left),
37             get_left_min(cur->right)));
38 }
39 Node* merge(Node *left_tree, Node *right_tree)
40 {
41     if (left_tree == nullptr)
42         return right_tree;
43     if (right_tree == nullptr)
44         return left_tree;
45
46     if (left_tree->priority < right_tree->priority)
47     {
48         right_tree->left = merge(left_tree, right_tree->left);
49         update(right_tree);
50         return right_tree;
51     }
52     else
53     {
54         left_tree->right = merge(left_tree->right, right_tree);
55         update(left_tree);
56         return left_tree;
57     }
58 }
59
60 pair<Node*, Node*> split(Node *cur, pair<int, int> key)
61 {
62     if (cur == nullptr)
63         return {nullptr, nullptr};
64
65     if (key > cur->key)
66     {
67         auto [left_split, right_split] = split(cur->right, key);
68         cur->right = left_split;
69         update(cur);
70         return {cur, right_split};
71     }
72     else
73     {
74         auto [left_split, right_split] = split(cur->left, key);
75         cur->left = right_split;
76         update(cur);
77         return {left_split, cur};
78     }
79 }
80
81 void remove_node(Node *&cur, pair<int, int> key)

```

```

82 {
83     if (cur == nullptr)
84         return;
85
86     if (cur->key == key)
87     {
88         cur = merge(cur->left, cur->right);
89         if (cur != nullptr)
90             update(cur);
91         return;
92     }
93
94     if (cur->key > key)
95         remove_node(cur->left, key);
96     else
97         remove_node(cur->right, key);
98
99     update(cur);
100 }
101 // snippet-end
102
103 void solve()
104 {
105 }
106 }
107
108 signed main()
109 {
110     // ios::sync_with_stdio(false);
111     // cout.tie(nullptr);
112     // cin.tie(nullptr);
113     int T = 1;
114     // cin >> T;
115     while (T--)
116         solve();
117     return 0;
118 }

```

2 图论

2.1 HLD

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;

```

```

6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct HLD
20 {
21     int n;
22     int id = 0;
23     vector<vector<int>> adj;
24
25     vector<int> fa;
26     vector<int> deep;
27     vector<int> siz;
28
29     vector<int> dfn;
30     vector<int> rev;
31     vector<int> top;
32
33     HLD(int _n) : n(_n)
34     {
35         adj.resize(n + 1);
36         fa.resize(n + 1, -1);
37         deep.resize(n + 1);
38         siz.resize(n + 1);
39
40         dfn.resize(n + 1);
41         rev.resize(n + 1);
42         top.resize(n + 1);
43     }
44
45     void build(int root = 1)
46     {
47         id = 0;
48         dfs1(root, -1, 0);
49         dfs2(root, root);
50     }
51
52     void add(int u, int v)

```

```

53     {
54         adj[u].push_back(v);
55         adj[v].push_back(u);
56     }
57
58     void dfs1(int u, int p, int d)
59     {
60         fa[u] = p;
61         deep[u] = d;
62         siz[u] = 1;
63         for (auto v : adj[u])
64         {
65             if (v == p)
66                 continue;
67             dfs1(v, u, d + 1);
68             if (siz[v] > siz[adj[u][0]])
69                 swap(v, adj[u][0]);
70             siz[u] += siz[v];
71         }
72     }
73
74     void dfs2(int u, int t)
75     {
76         top[u] = t;
77         dfn[u] = id++;
78         rev[dfn[u]] = u;
79         for (auto v : adj[u])
80         {
81             if (v == fa[u])
82                 continue;
83             dfs2(v, v);
84         }
85     }
86
87     int dist(int u, int v)
88     {
89         return deep[u] + deep[v] - 2 * deep[lca(u, v)];
90     }
91
92     int lca(int u, int v)
93     {
94         while (top[u] != top[v])
95         {
96             if (deep[top[u]] < deep[top[v]])
97                 swap(u, v);
98             u = fa[top[u]];
99         }
100         return deep[u] < deep[v] ? u : v;

```

```

101 }
102
103 int kth(int u, int k)
104 {
105     if (k < 0) return -1;
106     if (deep[u] < k) return -1;
107     while (u != -1)
108     {
109         int d = dfn[u] - dfn[top[u]];
110         if (k <= d) return rev[dfn[u] - k];
111
112         k -= d + 1;
113         u = fa[top[u]];
114     }
115
116     return -1;
117 }
118
119 bool is_anc(int u, int v)
120 {
121     return dfn[u] <= dfn[v] && dfn[v] < dfn[u] + siz[u];
122 }
123
124 vector<pair<int,int>> vtree(vector<int> nodes, int root = 1)
125 {
126     auto cmp = [&](int x, int y) { return dfn[x] < dfn[y]; };
127     sort(nodes.begin(), nodes.end(), cmp);
128     nodes.erase(unique(nodes.begin(), nodes.end()), nodes.end());
129
130     vector<int> all = nodes;
131     for (int i = 1; i < nodes.size(); i++) all.push_back(lca(nodes[i - 1], nodes
132 [i]));
133     sort(all.begin(), all.end(), cmp);
134     all.erase(unique(all.begin(), all.end()), all.end());
135
136     vector<pair<int,int>> edges;
137     vector<int> st;
138     for (int v : all)
139     {
140         if (st.empty())
141         {
142             st.push_back(v);
143             continue;
144         }
145         while (!st.empty() && !is_anc(st.back(), v))
146             st.pop_back();
147         edges.emplace_back(st.back(), v);
148         st.push_back(v);

```

```

148     }
149     return edges;
150 }
151 };
152 // snippet-end
153
154 void solve()
155 {
156 }
157 }
158
159 signed main()
160 {
161     // ios::sync_with_stdio(false);
162     // cout.tie(nullptr);
163     // cin.tie(nullptr);
164     int T = 1;
165     // cin >> T;
166     while (T--)
167         solve();
168     return 0;
169 }

```

2.2 HLD extend

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = int64_t;
4 using u64 = uint64_t;
5 using f64 = long double;
6 using i128 = __int128_t;
7 using u128 = __uint128_t;
8
9 const long double eps = 1e-12;
10 const i64 mod = 1e9 + 7;
11 const i64 INF = 1e18;
12 const int inf = 1e9;
13
14 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
15 auto rnd = [](i64 l, i64 r) { return (l <= r ? uniform_int_distribution<i64>(l, r)(
16     rng) : 0); };
17
18 // snippet-begin: HLD_extend
19 #define ls (node * 2 + 1)
20 #define rs (node * 2 + 2)
21
22 template<typename Info, typename Tag>
23 struct SegmentTree

```

```

23 {
24     int n;
25     vector<Info> tree;
26     vector<Tag> lazy;
27
28     SegmentTree() {}
29
30     SegmentTree(int _n) : n(_n)
31     {
32         init(vector<Info>(n, Info(0)));
33     }
34
35     template<typename T>
36     SegmentTree(const vector<T> &input) : n(input.size())
37     {
38         init(input);
39     }
40
41     template<typename T>
42     void init(const vector<T> &input)
43     {
44         n = input.size();
45         tree.resize(4 * n + 5, Info());
46         lazy.resize(4 * n + 5, Tag());
47         build(0, 0, n - 1, input);
48     }
49
50     template<typename T>
51     void build(int node, int start, int end, const vector<T> &input)
52     {
53         if (start == end)
54         {
55             tree[node] = input[start];
56         }
57         else
58         {
59             int mid = (start + end) / 2;
60             build(ls, start, mid, input);
61             build(rs, mid + 1, end, input);
62             tree[node] = tree[ls] + tree[rs];
63         }
64     }
65
66     void pushdown(int node)
67     {
68         if (lazy[node].empty()) return;
69
70         lazy[node].apply(tree[ls]);

```

```

71         lazy[node].apply(tree[rs]);
72
73         lazy[ls].merge(lazy[node]);
74         lazy[rs].merge(lazy[node]);
75
76         lazy[node] = Tag();
77     }
78
79     void update(int node, int start, int end, int l, int r, const Tag &val)
80     {
81         if (end < l || start > r) return;
82         if (l <= start && end <= r)
83         {
84             val.apply(tree[node]);
85             lazy[node].merge(val);
86             return;
87         }
88
89         pushdown(node);
90         int mid = (start + end) / 2;
91         if (l <= mid) update(ls, start, mid, l, r, val);
92         if (mid < r) update(rs, mid + 1, end, l, r, val);
93         tree[node] = tree[ls] + tree[rs];
94     }
95
96     void modify(int node, int start, int end, int pos, const Info &val)
97     {
98         if (start == end)
99         {
100             tree[node] = val;
101             return;
102         }
103
104         pushdown(node);
105         int mid = (start + end) / 2;
106         if (pos <= mid) modify(ls, start, mid, pos, val);
107         else if (pos > mid) modify(rs, mid + 1, end, pos, val);
108         tree[node] = tree[ls] + tree[rs];
109     }
110
111     Info query(int node, int start, int end, int l, int r)
112     {
113         if (l > end || r < start) return Info();
114         if (l <= start && end <= r) return tree[node];
115         pushdown(node);
116         int mid = (start + end) / 2;
117         return query(ls, start, mid, l, r) + query(rs, mid + 1, end, l, r);
118     }

```



```

119
120 void update(int l, int r, const Tag &val)
121 {
122     if (l > r) return;
123     update(0, 0, n - 1, l, r, val);
124 }
125
126 void modify(int pos, const Info &val)
127 {
128     modify(0, 0, n - 1, pos, val);
129 }
130
131 Info query(int l, int r)
132 {
133     if (l > r) return Info();
134     return query(0, 0, n - 1, l, r);
135 }
136 };
137
138 struct info
139 {
140     i64 mx = -INF;
141     i64 mn = INF;
142     i64 sum = 0;
143     i64 ssum = 0;
144     int len = 0;
145
146     info () : mx(-INF), mn(INF), sum(0), ssum(0), len(0) {};
147     info (i64 val) : mx(val), mn(val), sum(val), ssum(val * val), len(1) {};
148 };
149
150 info operator+(const info &l, const info &r)
151 {
152     info res;
153     res.mx = max(l.mx, r.mx);
154     res.mn = min(l.mn, r.mn);
155     res.sum = l.sum + r.sum;
156     res.ssum = l.ssum + r.ssum;
157     res.len = l.len + r.len;
158
159     return res;
160 }
161
162 // 区间加
163 struct tagAdd
164 {
165     i64 add = 0;
166

```

```

167 tagAdd() : add(0) {}
168 tagAdd(i64 _add) : add(_add) {}
169
170 bool empty() const
171 {
172     return add == 0;
173 }
174
175 void apply(info &a) const
176 {
177     i64 old = a.sum;
178
179     a.mx += add;
180     a.mn += add;
181     a.sum += add * a.len;
182     a.ssum += 2 * add * old + add * add * a.len;
183 }
184
185 void merge(const tagAdd &o)
186 {
187     if (o.empty())
188         return;
189
190     add += o.add;
191 }
192 };
193
194 // 区间赋值
195 struct tagAssign
196 {
197     bool has = false;
198     i64 val = 0;
199
200     tagAssign() : has(false), val(0) {};
201     tagAssign(i64 _val) : has(true), val(_val) {};
202
203     bool empty() const
204     {
205         return !has;
206     }
207
208     void apply(info &a) const
209     {
210         a.mx = val;
211         a.mn = val;
212         a.sum = val * a.len;
213         a.ssum = val * val * a.len;
214     }

```

```

215
216     void merge(const tagAssign &o)
217     {
218         if (!o.has)
219             return;
220
221         has = true;
222         val = o.val;
223     }
224 };
225
226 #undef ls
227 #undef rs
228
229 template<typename Tag>
230 struct HLD
231 {
232     int n;
233     int id;
234     int start;
235     int cap;
236     int use_edge;
237     vector<vector<int>> adj;
238
239     vector<int> fa;
240     vector<int> deep;
241     vector<int> siz;
242
243     vector<int> dfn;
244     vector<int> rev;
245     vector<int> top;
246
247     SegmentTree<info, Tag> tree;
248
249     HLD(int _n, int _start = 1) : n(_n), start(_start), cap(n + start)
250     {
251         adj.resize(cap);
252         fa.resize(cap, -1);
253         deep.resize(cap);
254         siz.resize(cap);
255
256         dfn.resize(cap);
257         rev.resize(cap);
258         top.resize(cap);
259     }
260
261     void build(int root)
262     {

```

```

263         id = 0;
264         dfs1(root, -1, 0);
265         dfs2(root, root);
266     }
267
268     template<typename T>
269     void init(vector<T> &input)
270     {
271         use_edge = 0;
272         vector<T> tmp(n, 0);
273         for (int i = start; i < cap; i++)
274             tmp[dfn[i]] = input[i];
275
276         tree.init(tmp);
277     }
278
279     template<typename T>
280     void init(vector<tuple<int, int, T>> &input)
281     {
282         use_edge = 1;
283         vector<T> tmp(n, 0);
284         for (auto [u, v, w] : input)
285         {
286             if (deep[u] > deep[v])
287                 tmp[dfn[u]] = w;
288             else
289                 tmp[dfn[v]] = w;
290         }
291
292         tree.init(tmp);
293     }
294
295     void add(int u, int v)
296     {
297         adj[u].push_back(v);
298         adj[v].push_back(u);
299     }
300
301     void dfs1(int u, int p, int d)
302     {
303         fa[u] = p;
304         deep[u] = d;
305         siz[u] = 1;
306         for (auto v : adj[u])
307         {
308             if (v == p)
309                 continue;
310             dfs1(v, u, d + 1);

```

```

311         if (siz[v] > siz[adj[u][0]])
312             swap(v, adj[u][0]);
313         siz[u] += siz[v];
314     }
315 }
316
317 void dfs2(int u, int t)
318 {
319     top[u] = t;
320     dfn[u] = id++;
321     rev[dfn[u]] = u;
322     for (auto v : adj[u])
323     {
324         if (v == fa[u])
325             continue;
326         dfs2(v, v);
327     }
328 }
329
330 int dist(int u, int v)
331 {
332     return deep[u] + deep[v] - 2 * deep[lca(u, v)];
333 }
334
335 int lca(int u, int v)
336 {
337     while (top[u] != top[v])
338     {
339         if (deep[top[u]] < deep[top[v]])
340             swap(u, v);
341         u = fa[top[u]];
342     }
343     return deep[u] < deep[v] ? u : v;
344 }
345
346 int kth(int u, int k)
347 {
348     if (k < 0) return -1;
349     if (deep[u] < k) return -1;
350     while (u != -1)
351     {
352         int d = dfn[u] - dfn[top[u]];
353         if (k <= d) return rev[dfn[u] - k];
354
355         k -= d + 1;
356         u = fa[top[u]];
357     }
358 }

```

```

359     return -1;
360 }
361
362 void update_path(int u, int v, const Tag &val)
363 {
364     while (top[u] != top[v])
365     {
366         if (deep[top[u]] < deep[top[v]])
367             swap(u, v);
368
369         int l = dfn[top[u]];
370         int r = dfn[u];
371         tree.update(l, r, val);
372         u = fa[top[u]];
373     }
374
375     int l = min(dfn[u], dfn[v]);
376     int r = max(dfn[u], dfn[v]);
377     tree.update(l + use_edge, r, val);
378 }
379
380 info query_path(int u, int v)
381 {
382     info res;
383     while (top[u] != top[v])
384     {
385         if (deep[top[u]] < deep[top[v]])
386             swap(u, v);
387
388         int l = dfn[top[u]];
389         int r = dfn[u];
390         res = res + tree.query(l, r);
391         u = fa[top[u]];
392     }
393
394     int l = min(dfn[u], dfn[v]);
395     int r = max(dfn[u], dfn[v]);
396     res = res + tree.query(l + use_edge, r);
397
398     return res;
399 }
400
401 void update_subtree(int u, const Tag &val)
402 {
403     int l = dfn[u];
404     int r = dfn[u] + siz[u] - 1;
405     tree.update(l + use_edge, r, val);
406 }

```

```

407     info query_subtree(int u)
408     {
409         int l = dfn[u];
410         int r = dfn[u] + siz[u] - 1;
411         return tree.query(l + use_edge, r);
412     }
413 };
414 // snippet-end
415
416 void solve()
417 {
418 }
419
420 int main()
421 {
422     // ios::sync_with_stdio(false);
423     // cout.tie(nullptr);
424     // cin.tie(nullptr);
425     int T = 1;
426     // cin >> T;
427     while (T--)
428         solve();
429     return 0;
430 }

```

2.3 HopcroftKarp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
17     rng) : 0); };
18 // snippet-begin:

```

```

19 struct HopcroftKarp
20 {
21     int n, m; // n: 左部顶点数, m: 右部顶点数
22     vector<vector<int>> adj; // 邻接表, 存储从左部到右部的边
23     vector<int> pair_u, pair_v; // 匹配数组。pair_u[u] = v, pair_v[v] = u
24     vector<int> dist; // BFS 中用于记录从左部未匹配点出发的距离
25
26     HopcroftKarp(int size_u, int size_v)
27     {
28         n = size_u;
29         m = size_v;
30         pair_u.resize(n + 1, 0);
31         pair_v.resize(m + 1, 0);
32         dist.resize(n + 1, 0);
33         adj.resize(n + 1);
34     }
35
36     void add(int u, int v)
37     {
38         adj[u].push_back(v);
39     }
40
41     /**
42      * @brief 通过 BFS 寻找增广路径, 并对左部顶点进行分层。
43      * @return 如果找到了至少一条增广路径, 返回 `true`; 否则返回 `false`。
44      */
45     bool bfs()
46     {
47         queue<int> q;
48         for (int u = 1; u <= n; u++)
49         {
50             if (pair_u[u] == 0) // 从所有未匹配的左部点开始
51             {
52                 dist[u] = 0;
53                 q.push(u);
54             }
55             else
56                 dist[u] = INT32_MAX;
57         }
58         dist[0] = INT32_MAX; // 虚拟NIL节点的距离设为无穷大
59         while (!q.empty())
60         {
61             int u = q.front();
62             q.pop();
63             if (dist[u] >= dist[0]) // 剪枝: 如果当前路径长度已超过已找到的增广路,
64                 则停止
65                 continue;
66             for (auto v : adj[u])

```

```

66     {
67         if (dist[pair_v[v]] == INT32_MAX) // 如果 v 的匹配点尚未被访问
68         {
69             dist[pair_v[v]] = dist[u] + 1;
70             q.push(pair_v[v]);
71         }
72     }
73 }
74 return dist[0] != INT32_MAX; // 如果NIL节点被访问, 说明找到了增广路
75 }
76
77 /**
78  * @brief 通过 DFS 在 BFS 构建的分层图上寻找一条增广路径。
79  * @param u 当前搜索的左部顶点 (1-based)。
80  * @return 如果从 u 出发找到了一条增广路径, 返回 `true`; 否则返回 `false`。
81  */
82 bool dfs(int u)
83 {
84     if (u == 0) // 到达虚拟NIL节点, 说明成功找到一条增广路径
85         return true;
86     for (auto v : adj[u])
87     {
88         if (dist[pair_v[v]] == dist[u] + 1) // 沿着分层图的边搜索
89         {
90             if (dfs(pair_v[v])) // 递归查找
91             {
92                 pair_u[u] = v;
93                 pair_v[v] = u;
94                 return true;
95             }
96         }
97     }
98     dist[u] = INT32_MAX; // 从 u 出发无法找到增广路, 将其距离设为无穷, 防止后续
99     访问
100     return false;
101 }
102
103 /**
104  * @brief 计算二分图的最大匹配数。
105  * @return 最大匹配数。
106  */
107 int max_matching()
108 {
109     int matching = 0;
110     while (bfs()) // 只要还能找到增广路
111     {
112         for (int u = 1; u <= n; u++)
113         {

```

```

113         if (pair_u[u] == 0) // 尝试为每个未匹配的左部点寻找增广路
114         {
115             if (dfs(u))
116                 matching++;
117         }
118     }
119     return matching;
120 }
121
122 /**
123  * @brief 获取最终的匹配结果。
124  * @return 一个向量 `pair_u`, 其中 `pair_u[i]` 表示与左部顶点 `i` 匹配的右部顶
125  * 点。
126  */
127 vector<int> get_matching()
128 {
129     return pair_u;
130 }
131 };
132 // snippet-end
133
134 void solve()
135 {
136 }
137
138 signed main()
139 {
140     // ios::sync_with_stdio(false);
141     // cout.tie(nullptr);
142     // cin.tie(nullptr);
143     int T = 1;
144     // cin >> T;
145     while (T--)
146         solve();
147     return 0;
148 }
149

```

2.4 LCA

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;

```

```

8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct LCA
20 {
21     int n, max_log;
22     vector<vector<int>> up;
23     vector<int> depth, roots;
24
25     LCA(int _n)
26     {
27         n = _n;
28         max_log = log2(n) + 1;
29         up.resize(n + 1, vector<int>(max_log + 1, 0));
30         roots.resize(n + 1, -1);
31         depth.resize(n + 1, -1);
32     }
33
34     void build(vector<vector<int>> &adj)
35     {
36         auto dfs = [&](auto dfs, int u, int p, int d, int r) -> void
37         {
38             up[u][0] = p;
39             depth[u] = d;
40             roots[u] = r;
41
42             for (int v : adj[u])
43             {
44                 if (v == p)
45                     continue;
46                 dfs(dfs, v, u, d + 1, r);
47             }
48         };
49
50         for (int i = 1; i <= n; i++)
51         {
52             if (depth[i] == -1)
53                 dfs(dfs, i, 0, 0, i);
54         }

```

```

55
56         for (int j = 1; j < max_log; j++)
57         {
58             for (int i = 1; i <= n; i++)
59             {
60                 if (up[i][j - 1] == 0)
61                     continue;
62
63                 up[i][j] = up[up[i][j - 1]][j - 1];
64             }
65         }
66     }
67
68     void build(int root, vector<vector<int>> &adj)
69     {
70         auto dfs = [&](auto dfs, int u, int p, int d, int r) -> void
71         {
72             up[u][0] = p;
73             depth[u] = d;
74             roots[u] = r;
75
76             for (int v : adj[u])
77             {
78                 if (v == p)
79                     continue;
80                 dfs(dfs, v, u, d + 1, r);
81             }
82         };
83         dfs(dfs, root, 0, 0, root);
84
85         for (int j = 1; j < max_log; j++)
86         {
87             for (int i = 1; i <= n; i++)
88             {
89                 if (up[i][j - 1] == 0)
90                     continue;
91
92                 up[i][j] = up[up[i][j - 1]][j - 1];
93             }
94         }
95     }
96
97     int query(int u, int v)
98     {
99         if (roots[u] != roots[v])
100             return -1;
101
102         if (depth[u] < depth[v])

```

```

103         swap(u, v);
104
105     for (int j = max_log - 1; j >= 0; j--)
106     {
107         if (depth[u] - (1ll << j) >= depth[v])
108             u = up[u][j];
109     }
110
111     if (u == v)
112         return u;
113
114     for (int j = max_log - 1; j >= 0; j--)
115     {
116         if (up[u][j] != up[v][j])
117         {
118             u = up[u][j];
119             v = up[v][j];
120         }
121     }
122
123     return up[u][0];
124 }
125 };
126 // snippet-end
127
128 void solve()
129 {
130 }
131 }
132
133 signed main()
134 {
135     // ios::sync_with_stdio(false);
136     // cout.tie(nullptr);
137     // cin.tie(nullptr);
138     int T = 1;
139     // cin >> T;
140     while (T--)
141         solve();
142     return 0;
143 }

```

2.5 TarjanEBCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;

```

```

5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
17     rng) : 0); };
18 // snippet-begin:
19 struct TarjanEBCC
20 {
21     int n, id = 0, ebcc_count = 0, timer = 0;
22     vector<vector<pair<int, int>>> adj;
23     vector<vector<int>> ebcc, nadj;
24     vector<int> dfn, low, bel, stk;
25     vector<bool> bridge, instk;
26
27     TarjanEBCC(int _n) : n(_n)
28     {
29         instk.resize(n + 1);
30         adj.resize(n + 1);
31         dfn.resize(n + 1);
32         low.resize(n + 1);
33         bel.resize(n + 1);
34         ebcc.resize(1);
35     }
36
37     void add(int u, int v)
38     {
39         id++;
40         adj[u].push_back({v, id});
41         adj[v].push_back({u, id});
42     }
43
44     void dfs(int u, int fid)
45     {
46         dfn[u] = low[u] = ++timer;
47         stk.push_back(u);
48         instk[u] = true;
49
50         for (auto [v, eid] : adj[u])
51

```

```

52     if (eid == fid) continue;
53
54     if (!dfn[v])
55     {
56         dfs(v, eid);
57         low[u] = min(low[u], low[v]);
58
59         if (low[v] > dfn[u])
60             bridge[eid] = true;
61     }
62     else if (instk[v])
63         low[u] = min(low[u], dfn[v]);
64 }
65
66 if (dfn[u] == low[u])
67 {
68     ebcc_count++;
69     ebcc.emplace_back();
70     while (true)
71     {
72         int x = stk.back();
73         stk.pop_back();
74         instk[x] = false;
75         bel[x] = ebcc_count;
76         ebcc.back().push_back(x);
77         if (x == u) break;
78     }
79 }
80 }
81
82 void run()
83 {
84     bridge.assign(id + 1, 0);
85     for (int i = 1; i <= n; i++)
86     {
87         if (!dfn[i])
88         {
89             dfs(i, -1);
90         }
91     }
92 }
93
94 void build()
95 {
96     nadj.assign(ebcc_count + 1, {});
97     for (int u = 1; u <= n; u++)
98     {
99         for (auto [v, id] : adj[u])

```

```

100         {
101             if (bridge[id])
102             {
103                 int x = bel[u];
104                 int y = bel[v];
105                 if (x > y)
106                 {
107                     nadj[x].push_back(y);
108                     nadj[y].push_back(x);
109                 }
110             }
111         }
112     }
113 }
114 };
115 // snippet-end
116
117 void solve()
118 {
119 }
120 }
121
122 signed main()
123 {
124     // ios::sync_with_stdio(false);
125     // cout.tie(nullptr);
126     // cin.tie(nullptr);
127     int T = 1;
128     // cin >> T;
129     while (T--)
130         solve();
131     return 0;
132 }

```

2.6 TarjanSCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;

```



```

13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct TarjanSCC
20 {
21     int n, scc_count = 0, timer = 0;
22     vector<vector<int>> adj, scc, nadj;
23     vector<int> dfn, low, bel, stk;
24     vector<bool> instk;
25
26     TarjanSCC(int _n) : n(_n)
27     {
28         instk.resize(n + 1);
29         adj.resize(n + 1);
30         dfn.resize(n + 1);
31         low.resize(n + 1);
32         bel.resize(n + 1);
33         scc.resize(1);
34     }
35
36     void add(int u, int v)
37     {
38         adj[u].push_back(v);
39     }
40
41     void dfs(int u)
42     {
43         dfn[u] = low[u] = ++timer;
44         stk.push_back(u);
45         instk[u] = 1;
46
47         for (auto v : adj[u])
48         {
49             if (!dfn[v])
50             {
51                 dfs(v);
52                 low[u] = min(low[u], low[v]);
53             }
54             else if (instk[v])
55                 low[u] = min(low[u], dfn[v]);
56         }
57
58         if (dfn[u] == low[u])
59         {

```

```

60             scc_count++;
61             scc.emplace_back();
62             while (true)
63             {
64                 int x = stk.back();
65                 stk.pop_back();
66                 instk[x] = false;
67                 bel[x] = scc_count;
68                 scc.back().push_back(x);
69                 if (x == u) break;
70             }
71         }
72     }
73
74     void run()
75     {
76         for (int i = 1; i <= n; i++)
77         {
78             if (!dfn[i])
79             {
80                 dfs(i);
81             }
82         }
83     }
84
85     void build()
86     {
87         nadj.resize(scc_count + 1);
88         vector<pair<int, int>> e;
89         for (int u = 1; u <= n; u++)
90         {
91             for (int v : adj[u])
92             {
93                 if (bel[u] != bel[v])
94                 {
95                     e.push_back({bel[u], bel[v]});
96                 }
97             }
98         }
99
100         sort(e.begin(), e.end());
101         e.erase(unique(e.begin(), e.end()), e.end());
102
103         for (auto [u, v] : e)
104             nadj[u].push_back(v);
105     }
106 };
107 // snippet-end

```

```

108
109 void solve()
110 {
111
112 }
113
114 signed main()
115 {
116     // ios::sync_with_stdio(false);
117     // cout.tie(nullptr);
118     // cin.tie(nullptr);
119     int T = 1;
120     // cin >> T;
121     while (T--)
122         solve();
123     return 0;
124 }

```

3 字符串

3.1 01tire

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct Trie
20 {
21     struct Node
22     {
23         array<int, 2> nex;
24         int cnt = 0;

```

```

25         int end = 0;
26
27         Node() { nex.fill(0); }
28     };
29
30     int mx = 31;
31     vector<Node> tree;
32     Trie(int n = 0)
33     {
34         tree.reserve(n * (mx + 1) + 1);
35         tree.emplace_back();
36     }
37
38     int newNode()
39     {
40         tree.emplace_back(Node());
41         return tree.size() - 1;
42     }
43
44     void insert(int x)
45     {
46         int p = 0;
47         for (int k = mx; k >= 0; k--)
48         {
49             int bit = (x >> k) & 1;
50             if (!tree[p].nex[bit])
51                 tree[p].nex[bit] = newNode();
52
53             p = tree[p].nex[bit];
54             tree[p].cnt++;
55         }
56
57         tree[p].end++;
58     }
59
60     int query(int x)
61     {
62         int p = 0, res = 0;
63         for (int k = mx; k >= 0; k--)
64         {
65             int bit = (x >> k) & 1;
66             if (tree[p].nex[bit ^ 1])
67             {
68                 res |= (1ll << k);
69                 p = tree[p].nex[bit ^ 1];
70             }
71             else
72                 p = tree[p].nex[bit];

```

```

73     }
74     return res;
75 }
76 };
77 // snippet-end
78
79 void solve()
80 {
81 }
82
83 }
84
85 signed main()
86 {
87     // ios::sync_with_stdio(false);
88     // cout.tie(nullptr);
89     // cin.tie(nullptr);
90     int T = 1;
91     // cin >> T;
92     while (T--)
93         solve();
94     return 0;
95 }

```

3.2 AhoCorasick

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using u32 = uint32_t;
4  using i64 = int64_t;
5  using u64 = uint64_t;
6  using f64 = long double;
7  using i128 = __int128_t;
8  using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
17     rng) : 0); };
18 // snippet-begin:
19 struct AhoCorasick
20 {
21     struct Node

```

```

22 {
23     array<int, 26> nex; // 子节点指针, 对于小写字母 'a'-'z'。
24     int cnt = 0; // 记录有多少个模式串经过此节点。
25     int end = 0; // 记录有多少个模式串在此节点结束。
26     int fail = 0; // fail 指针, 指向当前节点代表字符串的最长后缀所对应的节
27     点。
28     int link = 0; // 输出链 (字典序指针), 指向 fail 链上最近的、代表一个
29     完整模式串的节点。用于优化统计。
30     int occ = 0; // 出现次数, 在 query 时用于统计该节点代表的模式串在文本
31     串中的出现次数。
32
33     Node() { nex.fill(0); } // 构造时将所有子节点初始化为0 (不存在)。
34 };
35
36 vector<Node> tree; // 使用 vector 存储所有节点, 构成 Trie 树。tree[0] 是根节
37 点。
38 vector<int> endpos; // 记录每个模式串 (按插入顺序) 在 Trie 树中结尾节点的索引。
39 vector<int> bfs; // 存储 build 过程中节点的 BFS 遍历顺序, 用于 query 时按拓
40 扑序逆序更新 occ。
41
42 AhoCorasick(int n)
43 {
44     // 创建根节点。
45     tree.emplace_back();
46     // 根据模式串数量 n, 预分配 endpos 数组大小。
47     endpos.resize(n);
48 }
49
50 int newNode()
51 {
52     tree.emplace_back(Node());
53     return tree.size() - 1;
54 }
55
56 /**
57  * @brief 将一个模式串插入到 Trie 树中。
58  * @param s 要插入的模式串。
59  * @param id (可选) 该模式串的唯一ID, 用于在 `endpos` 中记录其末尾节点位置。
60  */
61 void insert(string &s, int id = 0)
62 {
63     int u = 0; // 从根节点开始
64     for (int i = 0; i < s.length(); i++)
65     {
66         int c = s[i] - 'a'; // 计算字符对应的索引
67         // 如果子节点不存在, 则创建一个新节点
68         if (!tree[u].nex[c])
69             tree[u].nex[c] = newNode();

```

```

65
66 // 移动到子节点
67 u = tree[u].nex[c];
68 tree[u].cnt++; // 经过该节点的模式串数量加一
69 }
70
71 tree[u].end++; // 在结尾节点, 标记一个模式串在此结束
72 endpos[id] = u; // 记录第 id 个模式串的结尾节点是 u
73 }
74
75 /**
76  * @brief 构建 AC 自动机。
77  * 核心是计算所有节点的 `fail` 指针, 并在此过程中“补全”Trie 图。
78  */
79 void build()
80 {
81     queue<int> q;
82     // 初始化队列, 将根节点的所有直接子节点入队
83     for (int c = 0; c < 26; c++)
84     {
85         if (tree[0].nex[c])
86         {
87             // 第一层节点的 fail 指针都指向根节点 0
88             tree[tree[0].nex[c]].fail = 0;
89             q.push(tree[0].nex[c]);
90         }
91     }
92
93     // BFS 遍历所有节点以计算 fail 指针
94     while (!q.empty())
95     {
96         int u = q.front();
97         q.pop();
98
99         // 记录 BFS 顺序, 用于后续查询
100         bfs.push_back(u);
101
102         for (int c = 0; c < 26; c++)
103         {
104             int v = tree[u].nex[c];
105             // 如果节点 u 存在字符 c 的子节点 v
106             if (v)
107             {
108                 // v 的 fail 指针是 u 的 fail 指针所指向的节点沿着相同字符 c 转
109                 // 移得到的节点
110                 tree[v].fail = tree[tree[u].fail].nex[c];
111                 // 计算 v 的输出链 link
112                 int to = tree[v].fail;

```

```

112 // 如果 v 的 fail 节点本身就是一个模式串的结尾, 则 link 指向它
113 if (tree[to].end > 0)
114     tree[v].link = to;
115 else // 否则, 继承 fail 节点的 link
116     tree[v].link = tree[to].link;
117
118 q.push(v);
119 }
120 else
121 {
122     // 如果节点 u 没有字符 c 的子节点, 则将该路径“补全”
123     // 直接连接到 u 的 fail 节点沿着 c 转移的路径上
124     tree[u].nex[c] = tree[tree[u].fail].nex[c];
125 }
126 }
127 }
128
129 /**
130  * @brief 在文本串 s 上执行匹配, 并统计每个模式串的出现次数。
131  * @param s 文本串。
132  */
133 void query(string &s)
134 {
135     int node = 0;
136     // 1. 遍历文本串 s, 在 AC 自动机上进行匹配
137     for (int i = 0; i < s.length(); i++)
138     {
139         int c = s[i] - 'a';
140         // 移动到下一个状态。因为 build 过程补全了路径, 所以可以直接转移
141         node = tree[node].nex[c];
142         // 匹配到的节点出现次数加一
143         tree[node].occ++;
144     }
145
146     // 2. 沿 fail 链反向更新出现次数
147     // 倒序遍历 BFS 序列 (相当于拓扑排序的逆序)
148     for (int i = bfs.size() - 1; i >= 0; i--)
149     {
150         int u = bfs[i];
151         // 将当前节点的出现次数累加到其 fail 指针指向的节点上
152         // 这样就保证了如果匹配到了 "abc", 那么 "bc" 和 "c" (如果它们是模式串)
153         // 也会被正确计数
154         tree[tree[u].fail].occ += tree[u].occ;
155     }
156 }
157
158 /**

```

```

159     * @brief 重置所有节点的出现次数 `occ`，以便进行下一次查询。
160     */
161     void reset()
162     {
163         // 遍历所有在 build 中访问过的节点（除了根节点）并重置 occ
164         for (auto u : bfs)
165             tree[u].occ = 0;
166         // 单独重置根节点的 occ
167         tree[0].occ = 0;
168     }
169 };
170 // snippet-end
171
172 void solve()
173 {
174 }
175 }
176
177 signed main()
178 {
179     // ios::sync_with_stdio(false);
180     // cout.tie(nullptr);
181     // cin.tie(nullptr);
182     int T = 1;
183     // cin >> T;
184     while (T--)
185         solve();
186     return 0;
187 }

```

3.3 PAM

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(

```

```

rng) : 0); });
17
18 // snippet-begin:
19 /**
20  * @brief 回文自动机 (PAM - Palindromic Automaton)，也称回文树。
21  * 用于在线性时间内处理字符串的所有回文子串信息。
22  */
23 struct PAM
24 {
25     /**
26     * @brief PAM 的节点结构。
27     */
28     struct node
29     {
30         array<int, 26> nex; // 子节点指针，nex[c] 指向在当前回文串两端加上字符 c 构
31         成的新回文串节点。
32         int fail = 0; // fail 指针，指向当前节点代表的回文串的最长回文后缀所对
33         应的节点。
34         int len = 0; // 当前节点代表的回文串的长度。
35         int end = 0; // 记录以当前节点代表的回文串为后缀的次数。调用 count()
36         后，变为在整个串中的出现次数。
37         int num = 0; // 当前节点代表的回文串所包含的回文后缀数量（包括自身）。
38     };
39
40     vector<node> tree; // 使用 vector 存储所有节点。
41     string s; // 存储构建 PAM 的字符串，为方便处理，下标从 1 开始。
42     int last; // 指向当前已处理字符串的最长回文后缀所对应的节点。
43
44     /**
45     * @brief 构造函数，初始化回文自动机。
46     * 创建两个根节点：0号节点（偶根，长度为0）和1号节点（奇根，长度为-1）。
47     */
48     PAM()
49     {
50         tree.emplace_back(); // 0号节点
51         tree.emplace_back(); // 1号节点
52         tree[0].len = 0;
53         tree[1].len = -1;
54
55         tree[0].fail = 1; // 偶根的 fail 指向奇根
56         tree[1].fail = 1; // 奇根的 fail 指向自身（或偶根，视实现而定）
57
58         s = " "; // 字符串下标从1开始，s[0]为占位符
59         last = 0; // 初始时，最长回文后缀是空串，对应0号节点
60     }
61
62     /**
63     * @brief 创建一个新节点。

```

```

61     * @return 返回新节点在节点数组中的索引。
62     */
63     int newNode()
64     {
65         tree.emplace_back();
66         return tree.size() - 1;
67     }
68
69     /**
70     * @brief 沿着 fail 链寻找一个合适的父节点。
71     *         该父节点 u 满足: s[i] + u的回文串 + s[i] 也是一个回文串。
72     * @param u 当前的 last 节点索引。
73     * @param i 新增字符 s[i] 的索引。
74     * @return 合适的父节点的索引。
75     */
76     int getFail(int u, int i)
77     {
78         // s[i - tree[u].len - 1] 是 u 对应回文串的前一个字符
79         while (s[i - tree[u].len - 1] != s[i])
80             u = tree[u].fail;
81         return u;
82     }
83
84     /**
85     * @brief 向自动机中插入一个新字符。
86     * @param ch 要插入的字符。
87     * @param i 字符在原字符串中的1-based索引。
88     */
89     void insert(char ch, int i)
90     {
91         s += ch;
92         int c = ch - 'a';
93         // 找到能扩展成新回文串的、当前串的最长回文后缀节点 u
94         int u = getFail(last, i);
95
96         // 如果这个新回文串不存在
97         if (!tree[u].nex[c])
98         {
99             int v = newNode(); // 创建新节点 v
100
101             tree[v].len = tree[u].len + 2;
102             // v 的 fail 指针是 u 的 fail 链上第一个能扩展成回文串的节点
103             tree[v].fail = tree[getFail(tree[u].fail, i)].nex[c];
104             tree[v].num = tree[tree[v].fail].num + 1;
105
106             tree[u].nex[c] = v;
107         }
108

```

```

109         // 更新 last 节点
110         last = tree[u].nex[c];
111         tree[last].end++;
112     }
113
114     /**
115     * @brief 统计每个本质不同回文子串在整个字符串中的出现次数。
116     *         必须在所有字符插入后调用。
117     *         利用 fail 树的性质，从叶节点向根节点累加 end 计数。
118     */
119     void count()
120     {
121         // 从后往前遍历节点（拓扑序的逆序），确保子节点的贡献先计算
122         for (int u = tree.size() - 1; u >= 2; u--)
123             tree[tree[u].fail].end += tree[u].end;
124     }
125 };
126 // snippet-end
127
128 void solve()
129 {
130
131 }
132
133 signed main()
134 {
135     // ios::sync_with_stdio(false);
136     // cout.tie(nullptr);
137     // cin.tie(nullptr);
138     int T = 1;
139     // cin >> T;
140     while (T--)
141         solve();
142     return 0;
143 }

```

3.4 StringHash

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = int64_t;
4 using u64 = uint64_t;
5 using f64 = long double;
6 using i128 = __int128_t;
7 using u128 = __uint128_t;
8
9 const long double eps = 1e-12;
10 const i64 mod = 1e9 + 7;

```

```

11 const i64 INF = 1e18;
12 const int inf = 1e9;
13
14 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
15 auto rnd = [](i64 l, i64 r) { return (1 <= r ? uniform_int_distribution<i64>(l, r)(
    rng) : 0); };
16
17 // snippet-begin:
18 struct StringHash
19 {
20     static constexpr u64 MOD = (1ull << 61) - 1;
21     inline static u64 BASE = rnd(MOD / 2, MOD - 2);
22     inline static vector<u64> P{1};
23     inline static int max_pow = 0;
24
25     struct Hash
26     {
27         u64 val = 0;
28         int len = 0;
29         Hash() = default;
30         Hash(u64 v, int l): val(v), len(l) {}
31
32         bool operator==(const Hash &o) const
33         {
34             return val == o.val && len == o.len;
35         }
36
37         Hash operator+(const Hash& rhs) const
38         {
39             ensure(rhs.len);
40             // return Hash(add(rhs.val, mul(val, P[rhs.len])), len + rhs.len);
41             return Hash((rhs.val + (u128)val * P[rhs.len]) % MOD, len);
42         }
43
44         bool operator<(const Hash &o) const
45         {
46             return (val < o.val) || (val == o.val && len < o.len);
47         }
48     };
49
50     vector<u64> h;
51     StringHash() = default;
52     StringHash(const string &s) { build(s); }
53
54     void build(const string &s)
55     {
56         int n = s.size();
57         ensure(n);

```

```

58         h.assign(n + 1, 0ull);
59         for (int i = 1; i <= n; ++i)
60         {
61             u64 v = (u64)(unsigned char)s[i - 1] + 1ull;
62             // h[i] = add(mul(h[i - 1], BASE), v);
63             h[i] = ((u128)h[i - 1] * BASE + v) % MOD;
64         }
65     }
66
67     Hash query(int l, int r)
68     {
69         if (r < l) return Hash(0, 0);
70         int m = r - l + 1;
71         ensure(m);
72         // return Hash(sub(h[r + 1], mul(h[l], P[m])), m);
73         return Hash(h[r + 1] - (u128)(MOD - h[l]) * P[m] % MOD, m);
74     }
75
76     Hash whole()
77     {
78         return Hash(h.back(), h.size() - 1);
79     }
80
81     // static u64 add(u64 a, u64 b)
82     // {
83     //     a += b;
84     //     if (a >= MOD) a -= MOD;
85     //     return a;
86     // }
87     // static u64 sub(u64 a, u64 b)
88     // {
89     //     return a >= b ? (a - b) : (a + MOD - b);
90     // }
91     // static u64 mul(u64 a, u64 b)
92     // {
93     //     u128 c = (u128)a * b;
94     //     u64 res = (u64)(c >> 61) + (u64)(c & MOD);
95     //     if (res >= MOD) res -= MOD;
96     //     return res;
97     // }
98     static void ensure(int m)
99     {
100         if (max_pow >= m) return;
101         P.resize(m + 1);
102         for (int i = max_pow + 1; i <= m; ++i) P[i] = (u128)P[i - 1] * BASE % MOD;
103         max_pow = m;
104     }
105 };

```

```

106 // snippet-end:
107
108 void solve()
109 {
110
111 }
112
113 int main()
114 {
115     // ios::sync_with_stdio(false);
116     // cout.tie(nullptr);
117     // cin.tie(nullptr);
118     int T = 1;
119     // cin >> T;
120     while (T--)
121         solve();
122     return 0;
123 }

```

3.5 Z

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 vector<int> Z(string &s)
20 {
21     int n = s.length();
22     vector<int> z(n);
23     int l = 0, r = 0;
24     for (int i = 1; i < n; i++)
25     {
26         if (i < r)

```

```

27         z[i] = min(z[l - i], r - i + 1);
28
29         while (i + z[i] - 1 < n && s[z[i]] == s[i + z[i] - 1])
30             z[i]++;
31
32         if (i + z[i] - 1 > r)
33         {
34             l = i;
35             r = i + z[i] - 1;
36         }
37     }
38
39     return z;
40 }
41 // snippet-end
42
43 void solve()
44 {
45
46 }
47
48 signed main()
49 {
50     // ios::sync_with_stdio(false);
51     // cout.tie(nullptr);
52     // cin.tie(nullptr);
53     int T = 1;
54     // cin >> T;
55     while (T--)
56         solve();
57     return 0;
58 }

```

3.6 manacher

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;

```



```

14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 /*
20 start = (i - p[i]) / 2;
21 end = (i + p[i]) / 2 - 1 = start + p[i] - 1;
22 [start, end] 代表原始字符串中以 (i) 或者 (i - 1 和 i) 为中心的回文串
23 */
24 vector<int> manacher(string &s)
25 {
26     int n = s.length();
27     vector<int> p(n);
28     int center = 0, r = 0;
29     for (int i = 0; i < n; i++)
30     {
31         int mr = 2 * center - i;
32         if (i < r)
33             p[i] = min(p[mr], r - i);
34
35         while (i - p[i] - 1 >= 0 && i + p[i] + 1 < n && s[i - p[i] - 1] == s[i + p[i]
36 ] + 1])
37             p[i]++;
38
39         if (i + p[i] - 1 > r)
40         {
41             center = i;
42             r = i + p[i] - 1;
43         }
44     }
45     return p;
46 }
47 // snippet-end
48
49 void solve()
50 {
51 }
52 }
53
54 signed main()
55 {
56     // ios::sync_with_stdio(false);
57     // cout.tie(nullptr);
58     // cin.tie(nullptr);
59     int T = 1;

```

```

60     // cin >> T;
61     while (T-->
62         solve();
63     return 0;
64 }

```

3.7 trie

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct Trie
20 {
21     struct Node
22     {
23         array<int, 26> nex;
24         int cnt = 0, end = 0;
25
26         Node() { nex.fill(0); }
27     };
28
29     vector<Node> tree;
30     Trie(int n = 0)
31     {
32         tree.reserve(n);
33         tree.emplace_back();
34     }
35
36     int newNode()
37     {
38         tree.emplace_back(Node());
39         return tree.size() - 1;

```

```

40 }
41
42 void insert(string s)
43 {
44     int p = 0;
45     for (int i = 0; i < s.length(); i++)
46     {
47         int c = s[i] - 'a';
48         if (!tree[p].nex[c])
49             tree[p].nex[c] = newNode();
50
51         p = tree[p].nex[c];
52         tree[p].cnt++;
53     }
54
55     tree[p].end++;
56 }
57
58 int find(string s)
59 {
60     int p = 0;
61     for (int i = 0; i < s.length(); i++)
62     {
63         int c = s[i] - 'a';
64         if (!tree[p].nex[c])
65             return 0;
66
67         p = tree[p].nex[c];
68     }
69
70     return tree[p].end;
71 }
72
73 };
74 // snippet-end
75
76 void solve()
77 {
78
79 }
80
81 signed main()
82 {
83     // ios::sync_with_stdio(false);
84     // cout.tie(nullptr);
85     // cin.tie(nullptr);
86     int T = 1;
87     // cin >> T;

```

```

88 while (T--)
89     solve();
90 return 0;
91 }

```

4 数学

4.1 BigNum

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 #ifndef BIG_ARITHMETIC_H
20 #define BIG_ARITHMETIC_H
21
22 struct BigInt
23 {
24     // 基数 B = 10^9
25     static const int BASE = 1e9;
26     // 基数的宽度，用于格式化输出
27     static const int WIDTH = 9;
28
29     vector<int> s; // 存储大数的"数位"
30     int sign;      // 符号: 1 为正或零, -1 为负
31
32     // ----- 构造与赋值 -----
33     BigInt() : sign(1) { s.push_back(0); }
34     BigInt(long long num) { *this = num; }
35     BigInt(const string& str) { *this = str; }
36
37 public:

```

```

38 BigInt& operator = (long long num)
39 {
40     s.clear();
41     sign = (num >= 0) ? 1 : -1;
42     if (num < 0) num = -num;
43     if (num == 0) s.push_back(0);
44     while (num > 0)
45     {
46         s.push_back(num % BASE);
47         num /= BASE;
48     }
49     return *this;
50 }
51
52 BigInt& operator = (const string& str)
53 {
54     s.clear();
55     int start = 0;
56     if (!str.empty() && str[0] == '-')
57     {
58         sign = -1;
59         start = 1;
60     }
61     else
62     {
63         sign = 1;
64     }
65
66     int len = str.length() - start;
67     if (len == 0)
68     {
69         s.push_back(0);
70         sign = 1;
71         return *this;
72     }
73
74     for (int i = len; i > 0; i -= WIDTH)
75     {
76         int t = 0;
77         int begin = max(0LL, (long long)i - WIDTH) + start;
78         for (int j = begin; j < i + start; j++)
79         {
80             t = t * 10 + str[j] - '0';
81         }
82         s.push_back(t);
83     }
84     normalize();
85     return *this;

```

```

86 }
87
88 // ----- 私有辅助函数 -----
89 void normalize()
90 {
91     while (s.size() > 1 && s.back() == 0) s.pop_back();
92     if (s.size() == 1 && s[0] == 0) sign = 1;
93 }
94
95 static int num_sign(long long n) { return (n < 0) ? -1 : 1; }
96
97 // 高效比较 BigInt 与 long long, 不创建临时对象
98 int compare_to_ll(long long num) const
99 {
100     if (this->sign != num_sign(num)) return this->sign > num_sign(num) ? 1 : -1;
101     if (s.empty() && num == 0) return 0;
102     if (s.empty()) return -1 * this->sign;
103
104     vector<int> num_s;
105     long long abs_num = abs(num);
106     if (abs_num == 0) num_s.push_back(0);
107     while(abs_num > 0) { num_s.push_back(abs_num % BASE); abs_num /= BASE; }
108
109     if (this->s.size() != num_s.size())
110         return (this->s.size() > num_s.size() ? 1 : -1) * this->sign;
111
112     for (int i = s.size() - 1; i >= 0; --i)
113     {
114         if (this->s[i] != num_s[i])
115             return (this->s[i] > num_s[i] ? 1 : -1) * this->sign;
116     }
117     return 0;
118 }
119
120 // ----- 公共辅助函数 -----
121 long long to_long_long() const
122 {
123     long long res = 0;
124     for (int i = s.size() - 1; i >= 0; --i) res = res * BASE + s[i];
125     return res * sign;
126 }
127
128 BigInt get_abs() const
129 {
130     BigInt res = *this;
131     res.sign = 1;
132     return res;
133 }

```

```

134
135 string to_string() const
136 {
137     stringstream ss;
138     ss << *this;
139     return ss.str();
140 }
141
142 BigInt pow(int n) const
143 {
144     BigInt res = 1, a = *this;
145     while(n > 0)
146     {
147         if(n & 1) res = res * a;
148         a = a * a;
149         n >>= 1;
150     }
151     return res;
152 }
153
154 static pair<BigInt, BigInt> div_mod(const BigInt& a, const BigInt& b)
155 {
156     if (b == 0) throw runtime_error("Division by zero");
157     if (a.get_abs() < b.get_abs()) return {BigInt(0), a};
158
159     BigInt q, r;
160     q.sign = a.sign * b.sign;
161
162     BigInt abs_a = a.get_abs();
163     BigInt abs_b = b.get_abs();
164
165     q.s.resize(abs_a.s.size());
166
167     for (int i = abs_a.s.size() - 1; i >= 0; i--)
168     {
169         r = r * BASE + abs_a.s[i];
170
171         // Binary search for the quotient digit
172         int l = 0, h = BigInt::BASE - 1, digit = 0;
173         while (l <= h)
174         {
175             int mid = l + (h - l) / 2;
176             if (abs_b * mid <= r)
177             {
178                 digit = mid;
179                 l = mid + 1;
180             } else
181                 h = mid - 1;

```

```

182     }
183     q.s[i] = digit;
184     r -= abs_b * digit;
185 }
186 q.normalize();
187 r.normalize();
188 if(r != 0) r.sign = a.sign; // 确保非零余数的符号正确
189
190 return {q, r};
191 }
192
193 // ----- 比较运算符 -----
194 bool operator < (const BigInt& other) const
195 {
196     if (sign != other.sign) return sign < other.sign;
197     if (s.size() != other.s.size()) return (s.size() < other.s.size()) ^ (sign
198 == -1);
199     for (int i = s.size() - 1; i >= 0; --i)
200         if (s[i] != other.s[i]) return (s[i] < other.s[i]) ^ (sign == -1);
201     return false;
202 }
203 bool operator > (const BigInt& other) const { return other < *this; }
204 bool operator <= (const BigInt& other) const { return !(*this > other); }
205 bool operator >= (const BigInt& other) const { return !(*this < other); }
206 bool operator == (const BigInt& other) const { return sign == other.sign && s ==
207 other.s; }
208 bool operator != (const BigInt& other) const { return !(*this == other); }
209
210 bool operator < (long long num) const { return compare_to_ll(num) < 0; }
211 bool operator > (long long num) const { return compare_to_ll(num) > 0; }
212 bool operator <= (long long num) const { return compare_to_ll(num) <= 0; }
213 bool operator >= (long long num) const { return compare_to_ll(num) >= 0; }
214 bool operator == (long long num) const { return compare_to_ll(num) == 0; }
215 bool operator != (long long num) const { return compare_to_ll(num) != 0; }
216
217 // ----- 算术: BigInt vs BigInt (成员函数) -----
218 BigInt operator + (const BigInt& other) const;
219 BigInt operator - (const BigInt& other) const;
220 BigInt operator * (const BigInt& other) const;
221 BigInt operator / (const BigInt& other) const;
222 BigInt operator % (const BigInt& other) const;
223
224 // ----- 算术: BigInt vs long long (高效成员函数) -----
225
226 BigInt operator + (long long num) const { return *this + BigInt(num); }
227 BigInt operator - (long long num) const { return *this - BigInt(num); }
228 BigInt operator * (long long num) const;
229 BigInt operator / (long long num) const;

```

```

227 long long operator % (long long num) const;
228
229 // ----- 复合赋值运算符 -----
230 BigInt& operator += (const BigInt& other) { return *this = *this + other; }
231 BigInt& operator -= (const BigInt& other) { return *this = *this - other; }
232 BigInt& operator *= (const BigInt& other) { return *this = *this * other; }
233 BigInt& operator /= (const BigInt& other) { return *this = *this / other; }
234 BigInt& operator %= (const BigInt& other) { return *this = *this % other; }
235
236 BigInt& operator += (long long num) { return *this = *this + num; }
237 BigInt& operator -= (long long num) { return *this = *this - num; }
238 BigInt& operator *= (long long num) { return *this = *this * num; }
239 BigInt& operator /= (long long num) { return *this = *this / num; }
240 // BigInt %= long long 没有意义, 因为结果是 long long
241
242 // ----- 一元运算符 -----
243 BigInt operator - () const { BigInt res = *this; if (res != 0) res.sign = -sign;
    return res; }
244
245 // ----- 友元函数 (用于 long long op BigInt) -----
246 friend BigInt operator + (long long a, const BigInt& b) { return b + a; }
247 friend BigInt operator - (long long a, const BigInt& b) { return -(b - a); }
248 friend BigInt operator * (long long a, const BigInt& b) { return b * a; }
249 friend BigInt operator / (long long a, const BigInt& b) { if (b == 0) throw
    runtime_error("Div by 0"); return (abs(a) < b.get_abs()) ? 0 : BigInt(a / b.
    to_long_long()); }
250 friend long long operator % (long long a, const BigInt& b) { if (b == 0) throw
    runtime_error("Mod by 0"); return (abs(a) < b.get_abs()) ? a : (a % b.
    to_long_long()); }
251
252 friend bool operator < (long long a, const BigInt& b) { return b > a; }
253 friend bool operator > (long long a, const BigInt& b) { return b < a; }
254 friend bool operator <= (long long a, const BigInt& b) { return b >= a; }
255 friend bool operator >= (long long a, const BigInt& b) { return b <= a; }
256 friend bool operator == (long long a, const BigInt& b) { return b == a; }
257 friend bool operator != (long long a, const BigInt& b) { return b != a; }
258
259 friend ostream& operator << (ostream& out, const BigInt& num);
260 friend istream& operator >> (istream& in, BigInt& num);
261 };
262
263 // ----- BigInt 函数实现 -----
264 BigInt BigInt::operator + (const BigInt& other) const
265 {
266     if (sign == other.sign)
267     {
268         BigInt res;
269         res.s.clear();

```

```

270         res.sign = sign;
271         long long carry = 0;
272         for (size_t i = 0; i < s.size() || i < other.s.size() || carry; ++i)
273         {
274             if (i < s.size()) carry += s[i];
275             if (i < other.s.size()) carry += other.s[i];
276             res.s.push_back(carry % BASE);
277             carry /= BASE;
278         }
279         res.normalize(); return res;
280     }
281     if (sign == -1) return other - (-(*this));
282     return *this - (-other);
283 }
284
285 BigInt BigInt::operator - (const BigInt& other) const
286 {
287     if (sign == other.sign)
288     {
289         if (this->get_abs() >= other.get_abs())
290         {
291             BigInt res;
292             res.s.clear();
293             res.sign = sign;
294             long long borrow = 0;
295             for (size_t i = 0; i < s.size(); ++i)
296             {
297                 long long current = s[i] - borrow;
298                 if (i < other.s.size()) current -= other.s[i];
299                 if (current < 0) { current += BASE; borrow = 1; }
300                 else { borrow = 0; }
301                 res.s.push_back(current);
302             }
303             res.normalize(); return res;
304         }
305         return -(other - *this);
306     }
307     if (sign == -1) return -(-(*this) + other);
308     return *this + (-other);
309 }
310
311 BigInt BigInt::operator * (const BigInt& other) const
312 {
313     BigInt res;
314     res.sign = sign * other.sign;
315     res.s.resize(s.size() + other.s.size());
316     for (size_t i = 0; i < s.size(); ++i)
317     {

```

```

318     long long carry = 0;
319     for (size_t j = 0; j < other.s.size() || carry > 0; ++j)
320     {
321         long long current = res.s[i + j] + carry;
322         if (j < other.s.size()) current += (long long)s[i] * other.s[j];
323         res.s[i + j] = current % BASE;
324         carry = current / BASE;
325     }
326 }
327 res.normalize(); return res;
328 }
329
330 BigInt BigInt::operator / (const BigInt& other) const
331 {
332     return div_mod(*this, other).first;
333 }
334
335 BigInt BigInt::operator % (const BigInt& other) const
336 {
337     return div_mod(*this, other).second;
338 }
339
340 BigInt BigInt::operator * (long long num) const
341 {
342     if (num == 0) return 0;
343     BigInt res = *this;
344     res.sign *= num_sign(num);
345     num = abs(num);
346
347     long long carry = 0;
348     for (size_t i = 0; i < res.s.size() || carry > 0; ++i)
349     {
350         if (i == res.s.size()) res.s.push_back(0);
351         long long current = res.s[i] * num + carry;
352         res.s[i] = current % BASE;
353         carry = current / BASE;
354     }
355     res.normalize(); return res;
356 }
357
358 BigInt BigInt::operator / (long long num) const
359 {
360     if (num == 0) throw runtime_error("Division by zero");
361     BigInt res; res.sign = this->sign;
362     if (num < 0) { res.sign *= -1; num = -num; }
363
364     res.s.resize(s.size());
365     long long rem = 0;

```

```

366     for (int i = s.size() - 1; i >= 0; --i)
367     {
368         long long current = rem * BASE + s[i];
369         res.s[i] = current / num;
370         rem = current % num;
371     }
372     res.normalize(); return res;
373 }
374
375 long long BigInt::operator % (long long num) const
376 {
377     if (num == 0) throw runtime_error("Modulo by zero");
378     num = abs(num);
379     long long rem = 0;
380     for (int i = s.size() - 1; i >= 0; --i)
381     {
382         rem = (rem * BASE + s[i]) % num;
383     }
384     return rem * this->sign;
385 }
386
387 ostream& operator << (ostream& out, const BigInt& num)
388 {
389     if (num.sign == -1) out << '-';
390     out << (num.s.empty() ? 0 : num.s.back());
391     for (int i = num.s.size() - 2; i >= 0; --i)
392     {
393         out << setfill('0') << setw(BigInt::WIDTH) << num.s[i];
394     }
395     return out;
396 }
397
398 istream& operator >> (istream& in, BigInt& num)
399 {
400     string str;
401     if (in >> str) num = str;
402     return in;
403 }
404
405 inline BigInt abs(const BigInt& num)
406 {
407     return num.get_abs();
408 }
409
410 #endif // BIG_ARITHMETIC_H
411
412 #ifndef BIG_DECIMAL_H
413 #define BIG_DECIMAL_H

```

```

414 struct BigDecimal
415 {
416     // 用于控制除法运算的额外精度，可根据需要调整
417     static const int DIVISION_PRECISION = 100;
418
419     BigInt value;    // 存储 scaled 后的整数部分
420     size_t precision; // 存储小数位数
421
422     // ----- 构造函数 -----
423     BigDecimal() : value(0), precision(0) {}
424     BigDecimal(long long num) : value(num), precision(0) {}
425     BigDecimal(const BigInt& v) : value(v), precision(0) {}
426     BigDecimal(const string& str) { *this = str; }
427
428 public:
429     // 私有构造函数，仅在内部使用，避免重复 normalize
430     BigDecimal(const BigInt& v, size_t p) : value(v), precision(p) {}
431
432     // ----- 赋值运算符 -----
433     BigDecimal& operator = (const string& str)
434     {
435         string s = str;
436         int s_sign = 1;
437         if (!s.empty() && s[0] == '-') { s_sign = -1; s = s.substr(1); }
438
439         size_t dot_pos = s.find('.');
440         if (dot_pos == string::npos)
441         {
442             value = BigInt(s);
443             precision = 0;
444         }
445         else
446         {
447             precision = s.length() - dot_pos - 1;
448             s.erase(dot_pos, 1);
449             value = BigInt(s);
450         }
451         value.sign = s_sign;
452         normalize();
453         return *this;
454     }
455
456     // ----- 私有辅助函数 -----
457     void normalize()
458     {
459         if (value == 0) { precision = 0; return; }
460         while (precision > 0 && value % 10 == 0)
461

```

```

462     {
463         value /= 10;
464         precision--;
465     }
466 }
467
468 // ----- 公共辅助函数 -----
469 string to_string() const
470 {
471     string s = value.get_abs().to_string();
472     string res = "";
473     if (value.sign == -1) res += '-';
474
475     if (precision == 0)
476     {
477         res += s;
478     }
479     else
480     {
481         if (s.length() <= precision)
482         {
483             res += "0.";
484             res += string(precision - s.length(), '0');
485             res += s;
486         }
487         else
488         {
489             res += s.substr(0, s.length() - precision) + "." + s.substr(s.length
490             () - precision);
491         }
492         return res;
493     }
494
495     BigDecimal get_abs() const
496     {
497         BigDecimal res = *this;
498         // 确保非零值才修改符号，零的符号总是1
499         if (res.value != 0)
500         {
501             res.value.sign = 1;
502         }
503         return res;
504     }
505
506     // ----- 核心算术: BigDecimal op BigDecimal -----
507     BigDecimal operator + (const BigDecimal& other) const
508     {

```

```

509     BigInt v1 = this->value;
510     BigInt v2 = other.value;
511     size_t target_precision = max(this->precision, other.precision);
512     if (this->precision < target_precision) v1 = v1 * BigInt(10).pow(
target_precision - this->precision);
513     if (other.precision < target_precision) v2 = v2 * BigInt(10).pow(
target_precision - other.precision);
514     BigDecimal res(v1 + v2, target_precision);
515     res.normalize();
516     return res;
517 }
518
519 BigDecimal operator - (const BigDecimal& other) const
520 {
521     BigInt v1 = this->value;
522     BigInt v2 = other.value;
523     size_t target_precision = max(this->precision, other.precision);
524     if (this->precision < target_precision) v1 = v1 * BigInt(10).pow(
target_precision - this->precision);
525     if (other.precision < target_precision) v2 = v2 * BigInt(10).pow(
target_precision - other.precision);
526     BigDecimal res(v1 - v2, target_precision);
527     res.normalize();
528     return res;
529 }
530
531 BigDecimal operator * (const BigDecimal& other) const
532 {
533     BigDecimal res(this->value * other.value, this->precision + other.precision);
534     res.normalize();
535     return res;
536 }
537
538 BigDecimal operator / (const BigDecimal& other) const
539 {
540     if (other.value == 0) throw runtime_error("BigDecimal division by zero");
541     BigInt dividend = this->value * BigInt(10).pow(other.precision +
DIVISION_PRECISION);
542     BigInt new_value = dividend / other.value;
543     size_t new_precision = this->precision + DIVISION_PRECISION;
544     BigDecimal res(new_value, new_precision);
545     res.normalize();
546     return res;
547 }
548
549 // 注意: Modulo (%) 对小数没有明确的通用定义, 因此不予实现.
550

```

```

551 // ----- 混合类型算术 (通过类型转换实现) -----
552 BigDecimal operator + (const BigInt& other) const { return *this + BigDecimal(
other); }
553 BigDecimal operator - (const BigInt& other) const { return *this - BigDecimal(
other); }
554 BigDecimal operator * (const BigInt& other) const { return *this * BigDecimal(
other); }
555 BigDecimal operator / (const BigInt& other) const { return *this / BigDecimal(
other); }
556
557 BigDecimal operator + (long long other) const { return *this + BigDecimal(other)
; }
558 BigDecimal operator - (long long other) const { return *this - BigDecimal(other)
; }
559 BigDecimal operator * (long long other) const { return *this * BigDecimal(other)
; }
560 BigDecimal operator / (long long other) const { return *this / BigDecimal(other)
; }
561
562 // ----- 复合赋值运算符 (完整版) -----
563 BigDecimal& operator += (const BigDecimal& other) { return *this = *this + other
; }
564 BigDecimal& operator -= (const BigDecimal& other) { return *this = *this - other
; }
565 BigDecimal& operator *= (const BigDecimal& other) { return *this = *this * other
; }
566 BigDecimal& operator /= (const BigDecimal& other) { return *this = *this / other
; }
567
568 BigDecimal& operator += (const BigInt& other) { return *this = *this + other; }
569 BigDecimal& operator -= (const BigInt& other) { return *this = *this - other; }
570 BigDecimal& operator *= (const BigInt& other) { return *this = *this * other; }
571 BigDecimal& operator /= (const BigInt& other) { return *this = *this / other; }
572
573 BigDecimal& operator += (long long other) { return *this = *this + other; }
574 BigDecimal& operator -= (long long other) { return *this = *this - other; }
575 BigDecimal& operator *= (long long other) { return *this = *this * other; }
576 BigDecimal& operator /= (long long other) { return *this = *this / other; }
577
578 // ----- 比较运算符 (完整版) -----
579 bool operator < (const BigDecimal& other) const;
580 bool operator > (const BigDecimal& other) const { return other < *this; }
581 bool operator <= (const BigDecimal& other) const { return !(*this > other); }
582 bool operator >= (const BigDecimal& other) const { return !(*this < other); }
583 bool operator == (const BigDecimal& other) const;
584 bool operator != (const BigDecimal& other) const { return !(*this == other); }
585
586 bool operator < (const BigInt& other) const { return *this < BigDecimal(other); }

```



```

}
587 bool operator > (const BigInt& other) const { return *this > BigDecimal(other);
}
588 bool operator <= (const BigInt& other) const { return *this <= BigDecimal(other)
; }
589 bool operator >= (const BigInt& other) const { return *this >= BigDecimal(other)
; }
590 bool operator == (const BigInt& other) const { return *this == BigDecimal(other)
; }
591 bool operator != (const BigInt& other) const { return *this != BigDecimal(other)
; }
592
593 bool operator < (long long other) const { return *this < BigDecimal(other); }
594 bool operator > (long long other) const { return *this > BigDecimal(other); }
595 bool operator <= (long long other) const { return *this <= BigDecimal(other); }
596 bool operator >= (long long other) const { return *this >= BigDecimal(other); }
597 bool operator == (long long other) const { return *this == BigDecimal(other); }
598 bool operator != (long long other) const { return *this != BigDecimal(other); }
599
600 // ----- 一元运算符 -----
601 BigDecimal operator - () const { BigDecimal res = *this; res.value = -res.value;
return res; }
602
603 // ----- 友元函数 (用于 外部类型 op BigDecimal)
-----
604 friend BigDecimal operator + (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) + b; }
605 friend BigDecimal operator - (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) - b; }
606 friend BigDecimal operator * (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) * b; }
607 friend BigDecimal operator / (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) / b; }
608
609 friend BigDecimal operator + (long long a, const BigDecimal& b) { return
BigDecimal(a) + b; }
610 friend BigDecimal operator - (long long a, const BigDecimal& b) { return
BigDecimal(a) - b; }
611 friend BigDecimal operator * (long long a, const BigDecimal& b) { return
BigDecimal(a) * b; }
612 friend BigDecimal operator / (long long a, const BigDecimal& b) { return
BigDecimal(a) / b; }
613
614 friend bool operator < (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) < b; }
615 friend bool operator > (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) > b; }
616 friend bool operator <= (const BigInt& a, const BigDecimal& b) { return

```

```

BigDecimal(a) <= b; }
617 friend bool operator >= (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) >= b; }
618 friend bool operator == (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) == b; }
619 friend bool operator != (const BigInt& a, const BigDecimal& b) { return
BigDecimal(a) != b; }
620
621 friend bool operator < (long long a, const BigDecimal& b) { return BigDecimal(a)
< b; }
622 friend bool operator > (long long a, const BigDecimal& b) { return BigDecimal(a)
> b; }
623 friend bool operator <= (long long a, const BigDecimal& b) { return BigDecimal(a)
<= b; }
624 friend bool operator >= (long long a, const BigDecimal& b) { return BigDecimal(a)
>= b; }
625 friend bool operator == (long long a, const BigDecimal& b) { return BigDecimal(a)
== b; }
626 friend bool operator != (long long a, const BigDecimal& b) { return BigDecimal(a)
!= b; }
627
628 friend ostream& operator << (ostream& out, const BigDecimal& num) { out << num.
to_string(); return out; }
629 friend istream& operator >> (istream& in, BigDecimal& num) { string s; if (in >>
s) num = s; return in; }
630 };
631
632 // ----- BigDecimal 函数实现 -----
633 bool BigDecimal::operator < (const BigDecimal& other) const
634 {
635     BigInt v1 = this->value, v2 = other.value;
636     size_t p1 = this->precision, p2 = other.precision;
637     size_t target_precision = max(p1, p2);
638     if (p1 < target_precision) v1 = v1 * BigInt(10).pow(target_precision - p1);
639     if (p2 < target_precision) v2 = v2 * BigInt(10).pow(target_precision - p2);
640     return v1 < v2;
641 }
642
643 bool BigDecimal::operator == (const BigDecimal& other) const
644 {
645     BigDecimal temp_a = *this; temp_a.normalize();
646     BigDecimal temp_b = other; temp_b.normalize();
647     return temp_a.value == temp_b.value && temp_a.precision == temp_b.precision;
648 }
649
650 inline BigDecimal abs(const BigDecimal& num)
651 {
652     return num.get_abs();

```

```

653 }
654
655 #endif // BIG_DECIMAL_H
656
657 BigDecimal calc_arctan(int d, int precision_digits)
658 {
659     // 1. 放大因子：多加几位精度以防误差
660     int margin = 10;
661     BigInt scale_factor = BigInt(10).pow(precision_digits + margin);
662
663     // 2. 初始项：(1/d) * scale_factor
664     BigInt term = scale_factor / d;
665
666     // 3. 累加和 (BigInt 类型)
667     BigInt total_sum = term;
668
669     BigInt d_squared = BigInt(d) * d;
670
671     for (long long k = 1; ; ++k)
672     {
673         // 4. 高效迭代：只用整数除法
674         // term_k = term_{k-1} / d^2
675         term = term / d_squared;
676
677         // 5. 如果项变得太小，无法影响整数部分，则停止
678         if (term == 0)
679             break;
680
681         // 6. 累加或累减
682         // full_term = (-1)^k * term / (2k+1)
683         if (k % 2 == 1) // k=1, 3, 5... (减)
684             total_sum -= term / (2 * k + 1);
685         else // k=2, 4, 6... (加)
686             total_sum += term / (2 * k + 1);
687     }
688
689     // 7. 一次性转换为 BigDecimal
690     return BigDecimal(total_sum, precision_digits + margin);
691 }
692
693 BigDecimal get_pi(int precision)
694 {
695     BigDecimal arctan5 = calc_arctan(5, precision);
696     BigDecimal arctan239 = calc_arctan(239, precision);
697     return (arctan5 * 4 - arctan239) * 4;
698 }
699
700 BigDecimal get_e(int precision_digits)

```

```

701 {
702     // 1. 放大因子，多加几位以防误差
703     int margin = 10;
704     BigInt scale_factor = BigInt(10).pow(precision_digits + margin);
705
706     // 2. 初始项 T_0 = scale_factor / 0! = scale_factor
707     BigInt term = scale_factor;
708
709     // 3. 累加和，初始为 T_0
710     BigInt total_sum = term;
711
712     // 4. 从 k=1 开始迭代
713     for (long long k = 1; ; ++k)
714     {
715         // 4a. 高效递推：T_k = T_{k-1} / k
716         term = term / k;
717
718         // 4b. 终止条件：当项小到无法影响整数和时停止
719         if (term == 0)
720             break;
721
722         // 4c. 累加当前项
723         total_sum += term;
724     }
725
726     // 5. 一次性转换为 BigDecimal
727     return BigDecimal(total_sum, precision_digits + margin);
728 }
729 // snippet-end
730
731 void solve()
732 {
733 }
734
735
736 signed main()
737 {
738     // ios::sync_with_stdio(false);
739     // cout.tie(nullptr);
740     // cin.tie(nullptr);
741     int T = 1;
742     // cin >> T;
743     while (T--)
744         solve();
745     return 0;
746 }

```

4.2 Comb

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct Comb
20 {
21     int max_n;
22     vector<int> fact;
23     vector<int> ifact;
24
25     Comb() : max_n(0)
26     {
27         fact.push_back(1);
28         ifact.push_back(1);
29     }
30
31     Comb(int n) : max_n(n)
32     {
33         extend_to(n);
34     }
35
36     void extend_to(int new_max_n)
37     {
38         if (new_max_n <= max_n) return;
39
40         int old_max_n = max_n;
41         max_n = new_max_n;
42
43         fact.resize(max_n + 1);
44         ifact.resize(max_n + 1);
45
```

```
46         for (int i = old_max_n + 1; i <= max_n; i++)
47             fact[i] = (1LL * fact[i - 1] * i) % mod;
48
49         ifact[max_n] = fast_pow(fact[max_n], mod - 2);
50         for (int i = max_n - 1; i > old_max_n; i--)
51             ifact[i] = (1LL * ifact[i + 1] * (i + 1)) % mod;
52     }
53
54     int fast_pow(int a, int b)
55     {
56         int res = 1;
57         a %= mod;
58         while (b)
59         {
60             if (b & 1)
61                 res = (1LL * res * a) % mod;
62             a = (1LL * a * a) % mod;
63             b >>= 1;
64         }
65         return res;
66     }
67
68     int inv(int x)
69     {
70         if (x > max_n) extend_to(x);
71         return fast_pow(x, mod - 2);
72     }
73
74     int C(int n, int m)
75     {
76         if (n < m || m < 0) return 0;
77
78         if (n > max_n)
79             extend_to(2 * n);
80
81         return (((1LL * fact[n] * ifact[m]) % mod) * ifact[n - m]) % mod;
82     }
83
84     int A(int n, int m)
85     {
86         if (n < m || m < 0) return 0;
87
88         if (n > max_n)
89             extend_to(2 * n);
90
91         return (1LL * fact[n] * ifact[n - m]) % mod;
92     }
93 } Comb;
```

```

94 // snippet-end
95
96 void solve()
97 {
98
99 }
100
101 signed main()
102 {
103     // ios::sync_with_stdio(false);
104     // cout.tie(nullptr);
105     // cin.tie(nullptr);
106     int T = 1;
107     // cin >> T;
108     while (T--)
109         solve();
110     return 0;
111 }

```

4.3 FFT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 using Complex = complex<double>;
20 const double PI = acos(-1.0);
21
22 struct FTT
23 {
24     vector<int> rev;
25     vector<Complex> roots {Complex(0, 0), Complex(1, 0)};
26     FTT() {}

```

```

27
28 /**
29  * @brief 执行快速傅里叶变换 (FFT) 或其逆变换 (IFFT)。
30  *         采用 Cooley-Tukey 算法，在原数组上进行变换 (in-place)。
31  * @param a 要变换的多项式系数向量 (复数形式)。其大小必须是2的幂。
32  * @param invert 一个布尔值，`false` 表示执行正向 FFT，`true` 表示执行逆向 IFFT
33  */
34 void dft(vector<Complex> &a, bool invert)
35 {
36     int n = a.size();
37
38     if (rev.size() != n)
39     {
40         rev.resize(n);
41         int k = __builtin_ctz(n) - 1;
42         for (int i = 0; i < n; i++)
43             rev[i] = (rev[i] >> 1) >> 1 | ((i & 1) << k);
44     }
45     for (int i = 0; i < n; i++)
46     {
47         if (rev[i] < i)
48         {
49             swap(a[i], a[rev[i]]);
50         }
51     }
52
53     if (roots.size() < n)
54     {
55         int k = __builtin_ctz(roots.size());
56         roots.resize(n);
57         while ((1 << k) < n)
58         {
59             double ang = PI / (1 << k);
60             Complex e(cos(ang), sin(ang));
61             for (int i = 1 << (k - 1); i < (1 << k); i++)
62             {
63                 roots[2 * i] = roots[i];
64                 roots[2 * i + 1] = roots[i] * e;
65             }
66             k++;
67         }
68     }
69
70     for (int len = 2; len <= n; len <= 1)
71     {
72         for (int i = 0; i < n; i += len)
73         {

```

```

74     for (int j = 0; j < len / 2; j++)
75     {
76         Complex w = roots[j + len / 2];
77         if (invert) w = conj(w);
78
79         Complex u = a[i + j];
80         Complex v = w * a[i + j + len / 2];
81         a[i + j] = u + v;
82         a[i + j + len / 2] = u - v;
83     }
84 }
85
86
87 if (invert)
88 {
89     for (auto &x : a)
90     {
91         x /= n;
92     }
93 }
94 }
95
96 /**
97  * @brief 使用 FFT 计算两个多项式的乘积（卷积）。
98  * @param a 第一个多项式 A(x) 的系数向量。
99  * @param b 第二个多项式 B(x) 的系数向量。
100  * @return 返回表示乘积多项式 C(x) = A(x) * B(x) 的系数向量。
101  * @note 对于小规模输入（结果次数小于128），会回退到 O(n^2) 的朴素乘法以避免 FFT
102 的常数开销。
103 */
104 vector<i64> mul(const vector<i64> &a, const vector<i64> &b)
105 {
106     int siz_a = a.size();
107     int siz_b = b.size();
108     int tot = siz_a + siz_b - 1;
109     if (tot <= 0) return {};
110
111     if (tot < 128)
112     {
113         vector<i64> c(tot, 0);
114         for (int i = 0; i < siz_a; i++)
115         {
116             for (int j = 0; j < siz_b; j++)
117             {
118                 c[i + j] += a[i] * b[j];
119             }
120         }
121         return c;

```

```

121     }
122
123     vector<Complex> fa(a.begin(), a.end());
124     vector<Complex> fb(b.begin(), b.end());
125
126     int n = 1;
127     while (n < tot) n <= 1;
128
129     fa.resize(n);
130     fb.resize(n);
131
132     dft(fa, false);
133     dft(fb, false);
134
135     for (int i = 0; i < n; i++)
136         fa[i] *= fb[i];
137
138     dft(fa, true);
139
140     vector<i64> res(n);
141     for (int i = 0; i < n; i++)
142         res[i] = round(fa[i].real());
143
144     res.resize(tot);
145     return res;
146 }
147
148 /**
149  * @brief 以可读的数学格式打印多项式。
150  * @param p 要打印的多项式的系数向量。例如 {4, 23, 22, 15} 会被打印为 "15x^3 +
151 22x^2 + 23x + 4"。
152  */
153 void print_poly(const vector<i64> &p)
154 {
155     bool first_term = true;
156     for (int i = p.size() - 1; i >= 0; --i)
157     {
158         if (p[i] != 0)
159         {
160             if (!first_term)
161                 cout << " + ";
162             first_term = false;
163
164             cout << p[i];
165             if (i > 1)
166                 cout << "x^" << i;
167             else if (i == 1)
168                 cout << "x";

```

```

168     }
169 }
170     if (first_term)
171         cout << 0;
172 }
173
174 } fft;
175 // snippet-end
176
177 void solve()
178 {
179     // Example usage:
180     vector<i64> p1 = {1, 2, 3}; // 3x^2 + 2x + 1
181     vector<i64> p2 = {4, 5};    // 5x + 4
182     vector<i64> p3 = fft.mul(p1, p2); // Should be 15x^3 + 22x^2 + 23x + 4
183     fft.print_poly(p3);
184     cout << endl;
185 }
186
187 signed main()
188 {
189     ios::sync_with_stdio(false);
190     cout.tie(nullptr);
191     cin.tie(nullptr);
192     int T = 1;
193     // cin >> T;
194     while (T--)
195         solve();
196     return 0;
197 }

```

4.4 FWT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

```

```

16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 i64 fast_pow(i64 a, i64 b)
20 {
21     i64 res = 1;
22     a %= mod;
23     while (b)
24     {
25         if (b & 1)
26             res = (1LL * res * a) % mod;
27
28         a = (1LL * a * a) % mod;
29         b >>= 1;
30     }
31     return res;
32 }
33
34 i64 inv(i64 x)
35 {
36     return fast_pow(x, mod - 2);
37 }
38
39 struct FWT
40 {
41     FWT() {}
42
43     static void OR(vector<i64> &a, int type)
44     {
45         int n = a.size();
46         for (int len = 2; len <= n; len <= 1)
47         {
48             int step = len / 2;
49             for (int i = 0; i < n; i += len)
50             {
51                 for (int j = 0; j < step; j++)
52                 {
53                     a[i + j + step] = (a[i + j + step] + type * a[i + j] + mod) %
54                     mod;
55                 }
56             }
57         }
58     }
59
60     static void AND(vector<i64> &a, int type)
61     {
62         int n = a.size();

```

```

62     for (int len = 2; len <= n; len <<= 1)
63     {
64         int step = len / 2;
65         for (int i = 0; i < n; i += len)
66         {
67             for (int j = step - 1; j >= 0; j--)
68             {
69                 a[i + j] = (a[i + j] + type * a[i + j + step] + mod) % mod;
70             }
71         }
72     }
73 }
74
75 static void XOR(vector<i64> &a, int type)
76 {
77     int n = a.size();
78     for (int len = 2; len <= n; len <<= 1)
79     {
80         int step = len / 2;
81         for (int i = 0; i < n; i += len)
82         {
83             for (int j = 0; j < step; j++)
84             {
85                 i64 u = a[i + j];
86                 i64 v = a[i + j + step];
87
88                 a[i + j] = (u + v) % mod;
89                 a[i + j + step] = ((u - v) % mod + mod) % mod;
90             }
91         }
92     }
93
94     if (type == -1)
95     {
96         i64 invN = inv(n);
97         for (auto &x : a)
98         {
99             x = (x * invN) % mod;
100         }
101     }
102 }
103
104 using Func = function<void(vector<i64>&, int)>;
105 vector<i64> work(const vector<i64> &a, const vector<i64> &b, Func op)
106 {
107     int tot = max(a.size(), b.size());
108     if (tot <= 0) return {};
109     int n = 1;

```

```

110     while (n < tot) n <<= 1;
111
112     vector<i64> fa(a);
113     vector<i64> fb(b);
114
115     fa.resize(n);
116     fb.resize(n);
117
118     op(fa, 1);
119     op(fb, 1);
120
121     for (int i = 0; i < n; i++)
122         fa[i] = (fa[i] * fb[i]) % mod;
123
124     op(fa, -1);
125     fa.resize(tot);
126     return fa;
127 }
128 } fwt;
129 // snippet-end
130
131 void solve()
132 {
133 }
134 }
135
136 signed main()
137 {
138     // ios::sync_with_stdio(false);
139     // cout.tie(nullptr);
140     // cin.tie(nullptr);
141     int T = 1;
142     // cin >> T;
143     while (T--)
144         solve();
145     return 0;
146 }

```

4.5 LinearBasis

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;

```

```

9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct LinearBasis
20 {
21     int bits;
22     vector<i64> basis;
23
24     LinearBasis (int _bits) : bits(_bits)
25     {
26         basis.resize(bits + 1);
27     }
28
29     bool insert(i64 x)
30     {
31         for (int i = bits - 1; i >= 0; i--)
32         {
33             if (!(x >> i & 1))
34                 continue;
35
36             if (basis[i])
37                 x ^= basis[i];
38             else
39             {
40                 basis[i] = x;
41                 return true;
42             }
43         }
44
45         return false;
46     }
47
48     bool exist(i64 x)
49     {
50         for (int i = bits - 1; i >= 0; i--)
51         {
52             if (!(x >> i & 1))
53                 continue;
54
55             x ^= basis[i];

```

```

56     }
57
58     return x == 0;
59 }
60
61 i64 queryMIN()
62 {
63     for (int i = 0; i < bits; i++)
64     {
65         if (basis[i] != 0)
66             return basis[i];
67     }
68
69     return 0;
70 }
71
72 i64 queryMAX()
73 {
74     i64 res = 0;
75     for (int i = bits - 1; i >= 0; i--)
76     {
77         if (basis[i] == 0)
78             continue;
79
80         if (((res >> i) & 1))
81             res ^= basis[i];
82     }
83
84     return res;
85 }
86 };
87 // snippet-end
88
89 void solve()
90 {
91
92 }
93
94 signed main()
95 {
96     // ios::sync_with_stdio(false);
97     // cout.tie(nullptr);
98     // cin.tie(nullptr);
99     int T = 1;
100    // cin >> T;
101    while (T--)
102        solve();
103    return 0;

```


104 }

4.6 LinearSieve

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (1 <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 struct LinearSieve
20 {
21     int n;
22     vector<int> minp;
23     vector<int> primes;
24     vector<int> phi;
25     vector<int> mu;
26     vector<int> tau, cnt; //约数个数
27
28     LinearSieve(int _n = 2e6 + 5, bool enable_phi = false, bool enable_mu = false,
29         bool enable_tau = false) : n(_n)
30     {
31         minp.resize(n + 1);
32         if (enable_phi)
33         {
34             phi.resize(n + 1);
35             phi[1] = 1;
36         }
37         if (enable_mu)
38         {
39             mu.resize(n + 1);
40             mu[1] = 1;
41         }
42         if (enable_tau)
43         {

```

```

43         tau.resize(n + 1);
44         cnt.resize(n + 1);
45         tau[1] = 1;
46     }
47
48     for (int i = 2; i <= n; i++)
49     {
50         if (minp[i] == 0)
51         {
52             minp[i] = i;
53             primes.push_back(i);
54
55             if (enable_phi) phi[i] = i - 1;
56             if (enable_mu) mu[i] = -1;
57             if (enable_tau) { tau[i] = 2, cnt[i] = 1; }
58         }
59         for (int p : primes)
60         {
61             i64 x = 1LL * i * p;
62             if (x > n) break;
63             minp[x] = p;
64
65             if (p == minp[i])
66             {
67                 if (enable_phi) phi[x] = 1LL * p * phi[i];
68                 if (enable_mu) mu[x] = 0;
69                 if (enable_tau)
70                 {
71                     cnt[x] = cnt[i] + 1;
72                     tau[x] = tau[i] / (cnt[i] + 1) * (cnt[x] + 1);
73                 }
74                 break;
75             }
76             else
77             {
78                 if (enable_phi) phi[x] = 1LL * (p - 1) * phi[i];
79                 if (enable_mu) mu[x] = -mu[i];
80                 if (enable_tau)
81                 {
82                     cnt[x] = 1;
83                     tau[x] = tau[i] * 2;
84                 }
85             }
86         }
87     }
88 }
89
90 map<i64, i64> factorize(i64 x)

```

```

91 {
92     map<i64, i64> facts;
93
94     if (x <= n)
95     {
96         while (x > 1)
97         {
98             int p = minp[x];
99             int count = 0;
100             while (x % p == 0)
101             {
102                 x /= p;
103                 count++;
104             }
105             facts[p] += count;
106         }
107
108         return facts;
109     }
110
111     for (int p : primes)
112     {
113         if (1LL * p * p > x) break;
114
115         if (x % p == 0)
116         {
117             int count = 0;
118             while (x % p == 0)
119             {
120                 x /= p;
121                 count++;
122             }
123             facts[p] += count;
124         }
125     }
126
127     if (x > 1) facts[x] = 1;
128
129     return facts;
130 }
131
132 bool is_prime(int x)
133 {
134     if (x < 2 || x > n) return false;
135     return minp[x] == x;
136 }
137 } LS(2e6 + 5, false, false, false);
138 // snippet-end

```

```

139
140 void solve()
141 {
142
143 }
144
145 signed main()
146 {
147     // ios::sync_with_stdio(false);
148     // cout.tie(nullptr);
149     // cin.tie(nullptr);
150     int T = 1;
151     // cin >> T;
152     while (T--)
153         solve();
154     return 0;
155 }

```

4.7 MTT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 constexpr i64 P1 = 998244353;
20 constexpr i64 P2 = 1004535809;
21 constexpr i64 P3 = 469762049;
22
23 i64 fast_pow(i64 a, i64 b, const i64 mod)
24 {
25     i64 res = 1;
26     a %= mod;
27     while (b)

```

```

28 {
29     if (b & 1)
30         res = ((i128)res * a) % mod;
31
32     a = ((i128)a * a) % mod;
33     b >>= 1;
34 }
35 return res;
36 }
37
38 i64 inv(i64 x, i64 mod)
39 {
40     return fast_pow(x, mod - 2, mod);
41 }
42
43 /**
44  * @brief 计算模意义下的二次剩余，即求解方程  $x^2 = a \pmod{p}$ 。
45  * 该函数实现了 Tonelli-Shanks 算法，并包含了针对特殊情况的优化。
46  *
47  * @param a 方程中的常数项 a。
48  * @param mod 模数 p，要求必须是一个奇素数。
49  * @return 如果方程有解，返回其中一个解 x。方程的另一个解是 mod - x。
50  * 如果方程无解，返回 -1。
51  * 如果 a = 0，返回 0。
52  */
53 i64 sqrt_mod(i64 a, i64 mod)
54 {
55     // 将 a 化为最小正整数
56     a %= mod;
57     if (a < 0) a += mod;
58
59     // ----- 特殊情况处理 -----
60     // a = 0，解为 0
61     if (a == 0) return 0;
62     // p = 2，解为 a 本身
63     if (mod == 2) return a;
64
65     // ----- 使用欧拉判别法检查解是否存在 -----
66     //  $(a/p) = a^{((p-1)/2)} \pmod{p}$ 
67     // 如果结果为 p-1 (即 -1)，则无解
68     if (fast_pow(a, (mod - 1) / 2, mod) == mod - 1)
69         return -1;
70
71     // ----- p = 3 (mod 4) 的简单情况 -----
72     //  $x = a^{((p+1)/4)} \pmod{p}$ 
73     if (mod % 4 == 3)
74         return fast_pow(a, (mod + 1) / 4, mod);
75

```

```

76 // ----- p = 1 (mod 4) 的 Tonelli-Shanks 算法 -----
77 // 1. 将 p-1 分解为  $Q * 2^S$ ，其中 Q 是奇数
78 i64 S = 0;
79 i64 Q = mod - 1;
80 while (Q % 2 == 0)
81 {
82     S++;
83     Q /= 2;
84 }
85 // 如果 S=1，那么  $p = Q*2+1$ ，Q为奇数， $p=2Q+1$ ，此时  $p\%4=3$ ，上面已处理
86 // 所以这里的  $S \geq 2$ 
87
88 // 2. 找到一个二次非剩余 n
89 i64 n = 2;
90 while (fast_pow(n, (mod - 1) / 2, mod) != mod - 1)
91     n++;
92
93 // 3. 初始化变量
94 i64 M = S;
95 i64 c = fast_pow(n, Q, mod); //  $c = n^Q \pmod{p}$ 
96 i64 t = fast_pow(a, Q, mod); //  $t = a^Q \pmod{p}$ 
97 i64 R = fast_pow(a, (Q + 1) / 2, mod); //  $R = a^{((Q+1)/2)} \pmod{p}$ 
98
99 // 4. 主循环
100 while (t != 1)
101 {
102     if (t == 0) return 0; // a 是 0 的情况
103
104     // 找到最小的 i > 0 使得  $t^{(2^i)} = 1 \pmod{p}$ 
105     i64 i = 0;
106     i64 temp_t = t;
107     while (temp_t != 1)
108     {
109         temp_t = (i128)temp_t * temp_t % mod;
110         i++;
111     }
112
113     // 理论上不会发生，除非输入p不是素数
114     if (i >= M) return -1;
115
116     // 计算  $b = c^{(2^{M-i-1})}$ 
117     i64 b_exp = 1LL << (M - i - 1); //  $2^{(M-i-1)}$ 
118     i64 b = fast_pow(c, b_exp, mod);
119
120     // 更新 M, c, t, R
121     M = i;
122     c = (i128)b * b % mod;
123     t = (i128)t * c % mod;

```

```

124     R = (i128)R * b % mod;
125 }
126
127 return R;
128 }
129
130 template<i64 mod>
131 struct NTT
132 {
133     int G = 3;
134     vector<int> rev;
135     vector<i64> roots = {0, 1};
136
137     i64 fast_pow(i64 a, i64 b)
138     {
139         i64 res = 1;
140         a %= mod;
141         while (b)
142         {
143             if (b & 1)
144                 res = ((i128)res * a) % mod;
145
146             a = ((i128)a * a) % mod;
147             b >>= 1;
148         }
149         return res;
150     }
151
152     i64 inv(i64 x)
153     {
154         return fast_pow(x, mod - 2);
155     }
156
157     /**
158      * @brief 执行正向NTT (DFT), 这是一个原地变换。
159      * @param a 多项式系数向量。
160      */
161     void dft(vector<i64> &a)
162     {
163         int n = a.size();
164         if (rev.size() != n)
165         {
166             int k = __builtin_ctz(n) - 1;
167             rev.resize(n);
168             for (int i = 0; i < n; i++)
169                 rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
170         }
171         for (int i = 0; i < n; i++)

```

```

172         if (rev[i] < i)
173             swap(a[i], a[rev[i]]);
174
175         if (roots.size() < n)
176         {
177             int k = __builtin_ctz(roots.size());
178             roots.resize(n);
179             while ((1 << k) < n)
180             {
181                 i64 e = fast_pow(G, (mod - 1) / (1LL << (k + 1)));
182                 for (int i = 1 << (k - 1); i < (1 << k); i++)
183                 {
184                     roots[2 * i] = roots[i];
185                     roots[2 * i + 1] = (roots[i] * e) % mod;
186                 }
187                 k++;
188             }
189         }
190
191         for (int len = 2; len <= n; len <= 1)
192         {
193             for (int i = 0; i < n; i += len)
194             {
195                 for (int j = 0; j < len / 2; j++)
196                 {
197                     i64 u = a[i + j];
198                     i64 v = (roots[j + len / 2] * a[i + j + len / 2]) % mod;
199                     a[i + j] = (u + v) % mod;
200                     a[i + j + len / 2] = (u - v + mod) % mod;
201                 }
202             }
203         }
204     }
205
206     /**
207      * @brief 执行逆向NTT (IDFT), 这是一个原地变换。
208      * @param a 经过DFT的点值表示向量。
209      */
210     void idft(vector<i64> &a)
211     {
212         int n = a.size();
213         reverse(a.begin() + 1, a.end());
214         dft(a);
215         i64 tmp = inv(n);
216         for (int i = 0; i < n; i++)
217             a[i] = (a[i] * tmp) % mod;
218     }
219

```

```

220 /**
221  * @brief 使用NTT计算两个多项式的乘积（卷积）。
222  * @param a 第一个多项式的系数。
223  * @param b 第二个多项式的系数。
224  * @return 结果多项式的系数。
225  */
226 vector<i64> mul(const vector<i64> &a, const vector<i64> &b)
227 {
228     int tot = a.size() + b.size() - 1;
229     if (tot <= 0) return {};
230
231     if (tot <= 128)
232     {
233         vector<i64> c(tot);
234         for (int i = 0; i < a.size(); i++)
235         {
236             for (int j = 0; j < b.size(); j++)
237             {
238                 c[i + j] = (c[i + j] + (i128)a[i] * b[j]) % mod;
239             }
240         }
241         return c;
242     }
243
244     int n = 1;
245     while (n < tot) n <<= 1;
246
247     vector<i64> fa(a);
248     vector<i64> fb(b);
249
250     fa.resize(n);
251     fb.resize(n);
252
253     dft(fa);
254     dft(fb);
255
256     for (int i = 0; i < n; i++)
257         fa[i] = (fa[i] * fb[i]) % mod;
258
259     idft(fa);
260     fa.resize(tot);
261     return fa;
262 }
263
264 /**
265  * @brief 使用牛顿迭代法，计算多项式 A(x) 的模 x^n 乘法逆元。
266  * 寻找一个多项式 B(x)，使得 A(x) * B(x) = 1 (mod x^n)。
267  * 此版本在频域（点值表示）中进行核心计算，以减少 NTT/INTT 调用次数。

```

```

268     * 迭代公式为 B_new = B_old * (2 - A * B_old)。
269
270     * @param a 输入多项式 A(x) 的系数向量。
271     * @param n 结果所需的精度，即返回的多项式的系数数量。
272     * @return 一个系数向量，表示逆元多项式 B(x) 的前 n 项系数。
273     */
274 vector<i64> inv_poly(const vector<i64> &a, int n)
275 {
276     assert(a.size() > 0 && a[0] != 0);
277
278     vector<i64> b = {inv(a[0], mod)};
279
280     int k = 1;
281     while (k < n)
282     {
283         int nk = k << 1;
284
285         int limit = 1;
286         while (limit < (nk << 1)) limit <<= 1;
287
288         vector<i64> tmp_a(a.begin(), a.begin() + min(nk, (int)a.size()));
289         tmp_a.resize(limit);
290         b.resize(limit);
291
292         dft(tmp_a);
293         dft(b);
294
295         for (int i = 0; i < limit; i++)
296         {
297             i64 term = (2 - (tmp_a[i] * b[i]) % mod + mod) % mod;
298             b[i] = (b[i] * term) % mod;
299         }
300
301         idft(b);
302
303         b.resize(nk);
304         k = nk;
305     }
306
307     b.resize(n);
308     return b;
309 }
310
311 /**
312  * @brief 计算多项式在模 mod 意义下的平方根。
313  * 采用牛顿迭代法，每一步迭代将解的精度（正确的系数项数）加倍。
314  * 迭代公式为 B_new = (B_old + A * B_old^-1) / 2。
315  * @param a 输入多项式 A(x) 的系数向量。

```

```

316 * @param n 结果多项式所需的项数（即精度，结果对  $x^n$  取模）。
317 * @return 一个向量，表示  $A(x)$  的平方根  $B(x)$  的前  $n$  项系数。
318 * 返回的解满足  $B(x)^2 = A(x) \pmod{x^n}$ 。
319 */
320 vector<i64> sqrt_poly(const vector<i64> &a, int n)
321 {
322     if (n == 0) return {};
323
324     vector<i64> b(1);
325     b[0] = sqrt_mod(a[0], mod);
326     b[0] = min(b[0], mod - b[0]);
327
328     assert(b[0] >= 0);
329
330     vector<i64> inv_b(1);
331     inv_b[0] = inv(b[0], mod);
332     i64 inv2 = inv(2, mod);
333
334     int k = 1;
335     while (k < n)
336     {
337         int nk = k << 1;
338         vector<i64> inv_b_k = inv_poly(b, nk);
339
340         vector<i64> tmp_a(a.begin(), a.begin() + min(nk, (int)a.size()));
341         auto term = mul(tmp_a, inv_b_k);
342
343         b.resize(nk);
344         for (int i = 0; i < nk; i++)
345         {
346             b[i] = (b[i] + term[i]) % mod;
347             b[i] = (b[i] * inv2) % mod;
348         }
349
350         k = nk;
351     }
352
353     b.resize(n);
354     return b;
355 }
356
357 };
358
359 struct MTT
360 {
361     NTT<P1> ntt1;
362     NTT<P2> ntt2;
363     NTT<P3> ntt3;

```

```

364
365 const i64 inv_p1_p2 = inv(P1 % P2, P2);
366 const i64 inv_p1p2_p3 = inv((i128)P1 * P2 % P3, P3);
367
368 /**
369 * @brief 在任意模数下计算两个多项式的乘积。
370 * @param a 第一个多项式的系数。
371 * @param b 第二个多项式的系数。
372 * @return 结果多项式的系数，模 `mod`。
373 */
374 vector<i64> mul(const vector<i64> &a, const vector<i64> &b)
375 {
376     auto c1 = ntt1.mul(a, b);
377     auto c2 = ntt2.mul(a, b);
378     auto c3 = ntt3.mul(a, b);
379
380     int n = c1.size();
381     vector<i64> c(n);
382
383     for (int i = 0; i < n; i++)
384     {
385         i64 k1 = (i128)(c2[i] - c1[i] + P2) % P2 * inv_p1_p2 % P2;
386         i128 c12 = (i128)k1 * P1 + c1[i];
387
388         i64 k2 = ((i128)c3[i] - c12 % P3 + P3) % P3 * inv_p1p2_p3 % P3;
389         i128 c123 = c12 + (i128)k2 * P1 * P2;
390
391         c[i] = c123 % mod;
392     }
393
394     return c;
395 }
396
397 /**
398 * @brief 在任意模数下计算多项式逆元。
399 * @param a 输入多项式  $A(x)$  的系数，要求  $a[0]$  非零。
400 * @param n 需要计算的逆元多项式的系数数量。
401 * @return 结果多项式  $B(x) = A(x)^{-1}$  的前  $n$  个系数，模 `mod`。
402 */
403 vector<i64> inv_poly(const vector<i64> &a, i64 n)
404 {
405     assert(a.size() > 0 && a[0] != 0);
406
407     vector<i64> b;
408     b.assign(1, inv(a[0], mod));
409
410     int k = 1;
411     while (k < n)

```

```

412     {
413         i64 nk = k << 1;
414         vector<i64> tmp1(a.begin(), a.begin() + min((i64)a.size(), nk));
415
416         auto tmp2 = mul(tmp1, b);
417         tmp2.resize(nk, 0);
418
419         for (int i = 0; i < nk; i++)
420             tmp2[i] = (mod - tmp2[i]) % mod;
421         tmp2[0] = (tmp2[0] + 2) % mod;
422
423         b = mul(b, tmp2);
424         b.resize(nk, 0);
425         k <= 1;
426     }
427
428     b.resize(n, 0);
429     return b;
430 }
431
432 } mtt;
433 // snippet-end
434
435 void solve()
436 {
437 }
438 }
439
440 signed main()
441 {
442     // ios::sync_with_stdio(false);
443     // cout.tie(nullptr);
444     // cin.tie(nullptr);
445     int T = 1;
446     // cin >> T;
447     while (T--)
448         solve();
449     return 0;
450 }

```

4.8 Matrix

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = int64_t;
4 using u64 = uint64_t;
5 using f64 = long double;
6 using i128 = __int128_t;

```

```

7 using u128 = __uint128_t;
8
9 const long double eps = 1e-12;
10 const i64 mod = 1e9 + 7;
11 const i64 INF = 1e18;
12 const int inf = 1e9;
13
14 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
15 auto rnd = [](i64 l, i64 r) { return (l <= r ? uniform_int_distribution<i64>(l, r)(
    rng) : 0); };
16
17 // snippet-begin:
18 struct Matrix
19 {
20     int n, m;
21     vector<i64> mt;
22
23     Matrix() {}
24     Matrix(int _n, int _m): n(_n), m(_m) { mt.resize(n * m, 0LL); }
25     Matrix(initializer_list<initializer_list<i64>> init) : n(init.size()), m(init.
        begin()->size()), mt(1LL * n * m, 0)
26     {
27         int i = 0;
28         for (auto &row : init)
29         {
30             for (auto x : row)
31                 mt[i++] = (x % mod + mod) % mod;
32         }
33     }
34
35     static Matrix identity(int n)
36     {
37         Matrix I(n, n);
38         for (int i = 0; i < n; i++) I[i][i] = 1;
39         return I;
40     }
41
42     i64* operator[](int i) { return mt.data() + 1LL * i * m; }
43     const i64* operator[](int i) const { return mt.data() + 1LL * i * m; }
44
45     friend Matrix operator*(const Matrix &l, const Matrix &r)
46     {
47         assert(l.m == r.n);
48         Matrix res(l.n, r.m);
49         for (int i = 0; i < l.n; i++)
50         {
51             for (int k = 0; k < l.m; k++)
52                 {

```

```

53         if (l[i][k] == 0) continue;
54         for (int j = 0; j < r.m; j++)
55         {
56             res[i][j] = (res[i][j] + (i128)l[i][k] * r[k][j]) % mod;
57         }
58     }
59 }
60 return res;
61 }
62
63 friend ostream& operator<<(ostream &os, const Matrix &o)
64 {
65     for(int i = 0; i < o.n; ++i)
66     {
67         for(int j = 0; j < o.m; ++j)
68         {
69             os << o[i][j] << " \n"[j == o.m - 1];
70         }
71     }
72     return os;
73 }
74 };
75
76 Matrix fast_pow(Matrix base, i64 b)
77 {
78     assert(base.n == base.m);
79     Matrix res = Matrix::identity(base.n);
80     while (b)
81     {
82         if (b & 1) res = res * base;
83         base = base * base;
84         b >>= 1;
85     }
86     return res;
87 }
88 // snippet-end:
89
90 void solve()
91 {
92 }
93 }
94
95 int main()
96 {
97     // ios::sync_with_stdio(false);
98     // cout.tie(nullptr);
99     // cin.tie(nullptr);
100     int T = 1;

```

```

101     // cin >> T;
102     while (T--)
103         solve();
104     return 0;
105 }

```

4.9 NTT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 template<i64 mod>
20 struct NTT
21 {
22     int G = 3;
23     vector<int> rev;
24     vector<i64> roots = {0, 1};
25
26     i64 fast_pow(i64 a, i64 b)
27     {
28         i64 res = 1;
29         a %= mod;
30         while (b)
31         {
32             if (b & 1)
33                 res = ((i128)res * a) % mod;
34
35             a = ((i128)a * a) % mod;
36             b >>= 1;
37         }
38         return res;
39     }

```



```

40
41 i64 inv(i64 x)
42 {
43     return fast_pow(x, mod - 2);
44 }
45
46 /**
47  * @brief 计算模意义下的二次剩余，即求解方程  $x^2 = a \pmod{p}$ 。
48  * 该函数实现了 Tonelli-Shanks 算法，并包含了针对特殊情况的优化。
49  *
50  * @param a 方程中的常数项 a。
51  * @param mod 模数 p，要求必须是一个奇素数。
52  * @return 如果方程有解，返回其中一个解 x。方程的另一个解是 mod - x。
53  * 如果方程无解，返回 -1。
54  * 如果 a = 0，返回 0。
55  */
56 i64 sqrt_mod(i64 a)
57 {
58     // 将 a 化为最小正整数
59     a %= mod;
60     if (a < 0) a += mod;
61
62     // ----- 特殊情况处理 -----
63     // a = 0，解为 0
64     if (a == 0) return 0;
65     // p = 2，解为 a 本身
66     if (mod == 2) return a;
67
68     // ----- 使用欧拉判别法检查解是否存在 -----
69     //  $(a/p) = a^{((p-1)/2)} \pmod{p}$ 
70     // 如果结果为 p-1 (即 -1)，则无解
71     if (fast_pow(a, (mod - 1) / 2) == mod - 1)
72         return -1;
73
74     // ----- p = 3 (mod 4) 的简单情况 -----
75     //  $x = a^{((p+1)/4)} \pmod{p}$ 
76     if (mod % 4 == 3)
77         return fast_pow(a, (mod + 1) / 4);
78
79     // ----- p = 1 (mod 4) 的 Tonelli-Shanks 算法 -----
80     // 1. 将 p-1 分解为  $Q * 2^S$ ，其中 Q 是奇数
81     i64 S = 0;
82     i64 Q = mod - 1;
83     while (Q % 2 == 0)
84     {
85         S++;
86         Q /= 2;
87     }

```

```

88
89 // 如果 S=1，那么 p = Q*2+1，Q为奇数，p=2Q+1，此时 p%4=3，上面已处理
90 // 所以这里的 S >= 2
91
92 // 2. 找到一个二次非剩余 n
93 i64 n = 2;
94 while (fast_pow(n, (mod - 1) / 2) != mod - 1)
95     n++;
96
97 // 3. 初始化变量
98 i64 M = S;
99 i64 c = fast_pow(n, Q); //  $c = n^Q \pmod{p}$ 
100 i64 t = fast_pow(a, Q); //  $t = a^Q \pmod{p}$ 
101 i64 R = fast_pow(a, (Q + 1) / 2); //  $R = a^{((Q+1)/2)} \pmod{p}$ 
102
103 // 4. 主循环
104 while (t != 1)
105 {
106     if (t == 0) return 0; // a 是 0 的情况
107
108     // 找到最小的 i > 0 使得  $t^{(2^i)} = 1 \pmod{p}$ 
109     i64 i = 0;
110     i64 temp_t = t;
111     while (temp_t != 1)
112     {
113         temp_t = (i128)temp_t * temp_t % mod;
114         i++;
115     }
116
117     // 理论上不会发生，除非输入p不是素数
118     if (i >= M) return -1;
119
120     // 计算  $b = c^{(2^{M-i-1})}$ 
121     i64 b_exp = 1LL << (M - i - 1); //  $2^{(M-i-1)}$ 
122     i64 b = fast_pow(c, b_exp);
123
124     // 更新 M, c, t, R
125     M = i;
126     c = (i128)b * b % mod;
127     t = (i128)t * c % mod;
128     R = (i128)R * b % mod;
129 }
130
131 return R;
132 }
133
134 /**
135  * @brief 执行正向NTT (DFT)，这是一个原地变换。
136  * @param a 多项式系数向量。

```

```

136 */
137 void dft(vector<i64> &a)
138 {
139     int n = a.size();
140     if (rev.size() != n)
141     {
142         int k = __builtin_ctz(n) - 1;
143         rev.resize(n);
144         for (int i = 0; i < n; i++)
145             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
146     }
147     for (int i = 0; i < n; i++)
148         if (rev[i] < i)
149             swap(a[i], a[rev[i]]);
150
151     if (roots.size() < n)
152     {
153         int k = __builtin_ctz(roots.size());
154         roots.resize(n);
155         while ((1 << k) < n)
156         {
157             i64 e = fast_pow(G, (mod - 1) / (1LL << (k + 1)));
158             for (int i = 1 << (k - 1); i < (1 << k); i++)
159             {
160                 roots[2 * i] = roots[i];
161                 roots[2 * i + 1] = (roots[i] * e) % mod;
162             }
163             k++;
164         }
165     }
166
167     for (int len = 2; len <= n; len <<= 1)
168     {
169         for (int i = 0; i < n; i += len)
170         {
171             for (int j = 0; j < len / 2; j++)
172             {
173                 i64 u = a[i + j];
174                 i64 v = (roots[j + len / 2] * a[i + j + len / 2]) % mod;
175                 a[i + j] = (u + v) % mod;
176                 a[i + j + len / 2] = (u - v + mod) % mod;
177             }
178         }
179     }
180 }
181
182 /**
183  * @brief 执行逆向NTT (IDFT), 这是一个原地变换。

```

```

184  * @param a 经过DFT的点值表示向量。
185  */
186 void idft(vector<i64> &a)
187 {
188     int n = a.size();
189     reverse(a.begin() + 1, a.end());
190     dft(a);
191     i64 tmp = inv(n);
192     for (int i = 0; i < n; i++)
193         a[i] = (a[i] * tmp) % mod;
194 }
195
196 /**
197  * @brief 使用NTT计算两个多项式的乘积 (卷积)。
198  * @param a 第一个多项式的系数。
199  * @param b 第二个多项式的系数。
200  * @return 结果多项式的系数。
201  */
202 vector<i64> mul(const vector<i64> &a, const vector<i64> &b)
203 {
204     int tot = a.size() + b.size() - 1;
205     if (tot <= 0) return {};
206
207     if (tot <= 128)
208     {
209         vector<i64> c(tot);
210         for (int i = 0; i < a.size(); i++)
211         {
212             for (int j = 0; j < b.size(); j++)
213             {
214                 c[i + j] = (c[i + j] + (i128)a[i] * b[j]) % mod;
215             }
216         }
217         return c;
218     }
219
220     int n = 1;
221     while (n < tot) n <<= 1;
222
223     vector<i64> fa(a);
224     vector<i64> fb(b);
225
226     fa.resize(n);
227     fb.resize(n);
228
229     dft(fa);
230     dft(fb);
231

```

```

232     for (int i = 0; i < n; i++)
233         fa[i] = (fa[i] * fb[i]) % mod;
234
235     idft(fa);
236     fa.resize(tot);
237     return fa;
238 }
239
240 /**
241  * @brief 使用牛顿迭代法，计算多项式  $A(x)$  的模  $x^n$  乘法逆元。
242  *         寻找一个多项式  $B(x)$ ，使得  $A(x) * B(x) = 1 \pmod{x^n}$ 。
243  *         此版本在频域（点值表示）中进行核心计算，以减少 NTT/INTT 调用次数。
244  *         迭代公式为  $B_{\text{new}} = B_{\text{old}} * (2 - A * B_{\text{old}})$ 。
245  *
246  * @param a 输入多项式  $A(x)$  的系数向量。
247  * @param n 结果所需的精度，即返回的多项式的系数数量。
248  * @return 一个系数向量，表示逆元多项式  $B(x)$  的前  $n$  项系数。
249  */
250 vector<i64> inv_poly(const vector<i64> &a, int n)
251 {
252     assert(a.size() > 0 && a[0] != 0);
253
254     vector<i64> b = {inv(a[0], mod)};
255
256     int k = 1;
257     while (k < n)
258     {
259         int nk = k << 1;
260
261         int limit = 1;
262         while (limit < (nk << 1)) limit <= 1;
263
264         vector<i64> tmp_a(a.begin(), a.begin() + min(nk, (int)a.size()));
265         tmp_a.resize(limit);
266         b.resize(limit);
267
268         dft(tmp_a);
269         dft(b);
270
271         for (int i = 0; i < limit; i++)
272         {
273             i64 term = (2 - (tmp_a[i] * b[i]) % mod + mod) % mod;
274             b[i] = (b[i] * term) % mod;
275         }
276
277         idft(b);
278
279         b.resize(nk);

```

```

280         k = nk;
281     }
282
283     b.resize(n);
284     return b;
285 }
286
287 /**
288  * @brief 计算多项式在模 mod 意义下的平方根。
289  *         采用牛顿迭代法，每一步迭代将解的精度（正确的系数项数）加倍。
290  *         迭代公式为  $B_{\text{new}} = (B_{\text{old}} + A * B_{\text{old}}^{-1}) / 2$ 。
291  * @param a 输入多项式  $A(x)$  的系数向量。
292  * @param n 结果多项式所需的项数（即精度，结果对  $x^n$  取模）。
293  * @return 一个向量，表示  $A(x)$  的平方根  $B(x)$  的前  $n$  项系数。
294  *         返回的解满足  $B(x)^2 = A(x) \pmod{x^n}$ 。
295  */
296 vector<i64> sqrt_poly(const vector<i64> &a, int n)
297 {
298     if (n == 0) return {};
299
300     vector<i64> b(1);
301     b[0] = sqrt_mod(a[0], mod);
302     b[0] = min(b[0], mod - b[0]);
303
304     assert(b[0] >= 0);
305
306     vector<i64> inv_b(1);
307     inv_b[0] = inv(b[0], mod);
308     i64 inv2 = inv(2, mod);
309
310     int k = 1;
311     while (k < n)
312     {
313         int nk = k << 1;
314         vector<i64> inv_b_k = inv_poly(b, nk);
315
316         vector<i64> tmp_a(a.begin(), a.begin() + min(nk, (int)a.size()));
317         auto term = mul(tmp_a, inv_b_k);
318
319         b.resize(nk);
320         for (int i = 0; i < nk; i++)
321         {
322             b[i] = (b[i] + term[i]) % mod;
323             b[i] = (b[i] * inv2) % mod;
324         }
325
326         k = nk;
327     }

```

```

328         b.resize(n);
329         return b;
330     }
331 }
332 };
333 // snippet-end
334
335 void solve()
336 {
337 }
338 }
339 }
340
341 signed main()
342 {
343     // ios::sync_with_stdio(false);
344     // cout.tie(nullptr);
345     // cin.tie(nullptr);
346     int T = 1;
347     // cin >> T;
348     while (T--)
349         solve();
350     return 0;
351 }

```

4.10 fast pow

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;
11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 i64 fast_pow(i64 a, i64 b)
20 {

```

```

21     i64 res = 1;
22     a %= mod;
23     while (b)
24     {
25         if (b & 1)
26             res = (1LL * res * a) % mod;
27
28         a = (1LL * a * a) % mod;
29         b >>= 1;
30     }
31     return res;
32 }
33
34 i64 inv(i64 x)
35 {
36     return fast_pow(x, mod - 2);
37 }
38 // snippet-end
39
40 void solve()
41 {
42 }
43 }
44
45 signed main()
46 {
47     // ios::sync_with_stdio(false);
48     // cout.tie(nullptr);
49     // cin.tie(nullptr);
50     int T = 1;
51     // cin >> T;
52     while (T--)
53         solve();
54     return 0;
55 }

```

4.11 sqrt mod

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using u32 = uint32_t;
4 using i64 = int64_t;
5 using u64 = uint64_t;
6 using f64 = long double;
7 using i128 = __int128_t;
8 using u128 = __uint128_t;
9
10 const long double eps = 1e-12;

```

```

11 const i64 mod = 1e9 + 7;
12 const i64 INF = 1e18;
13 const int inf = 1e9;
14
15 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16 auto rnd = [](u64 l, u64 r) { return (l <= r ? uniform_int_distribution<u64>(l, r)(
    rng) : 0); };
17
18 // snippet-begin:
19 i64 fast_pow(i64 a, i64 b, i64 mod)
20 {
21     i64 res = 1;
22     a %= mod;
23     while (b)
24     {
25         if (b & 1)
26             res = (1LL * res * a) % mod;
27
28         a = (1LL * a * a) % mod;
29         b >>= 1;
30     }
31     return res;
32 }
33
34 i64 inv(i64 x, i64 mod)
35 {
36     return fast_pow(x, mod - 2, mod);
37 }
38
39 /**
40  * @brief 计算模意义下的二次剩余, 即求解方程  $x^2 = a \pmod{p}$ 。
41  * 该函数实现了 Tonelli-Shanks 算法, 并包含了针对特殊情况的优化。
42  *
43  * @param a 方程中的常数项 a。
44  * @param mod 模数 p, 要求必须是一个奇素数。
45  * @return 如果方程有解, 返回其中一个解 x。方程的另一个解是 mod - x。
46  * 如果方程无解, 返回 -1。
47  * 如果 a = 0, 返回 0。
48  */
49 i64 sqrt_mod(i64 a, i64 mod)
50 {
51     // 将 a 化为最小正整数
52     a %= mod;
53     if (a < 0) a += mod;
54
55     // ----- 特殊情况处理 -----
56     // a = 0, 解为 0
57     if (a == 0) return 0;

```

```

58 // p = 2, 解为 a 本身
59 if (mod == 2) return a;
60
61 // ----- 使用欧拉判别法检查解是否存在 -----
62 //  $(a/p) = a^{(p-1)/2} \pmod{p}$ 
63 // 如果结果为 p-1 (即 -1), 则无解
64 if (fast_pow(a, (mod - 1) / 2, mod) == mod - 1)
65     return -1;
66
67 // ----- p = 3 (mod 4) 的简单情况 -----
68 //  $x = a^{(p+1)/4} \pmod{p}$ 
69 if (mod % 4 == 3)
70     return fast_pow(a, (mod + 1) / 4, mod);
71
72 // ----- p = 1 (mod 4) 的 Tonelli-Shanks 算法 -----
73 // 1. 将 p-1 分解为  $Q * 2^S$ , 其中 Q 是奇数
74 i64 S = 0;
75 i64 Q = mod - 1;
76 while (Q % 2 == 0)
77 {
78     S++;
79     Q /= 2;
80 }
81 // 如果 S=1, 那么  $p = Q*2+1$ , Q为奇数,  $p=2Q+1$ , 此时  $p\%4=3$ , 上面已处理
82 // 所以这里的 S >= 2
83
84 // 2. 找到一个二次非剩余 n
85 i64 n = 2;
86 while (fast_pow(n, (mod - 1) / 2, mod) != mod - 1)
87     n++;
88
89 // 3. 初始化变量
90 i64 M = S;
91 i64 c = fast_pow(n, Q, mod); //  $c = n^Q \pmod{p}$ 
92 i64 t = fast_pow(a, Q, mod); //  $t = a^Q \pmod{p}$ 
93 i64 R = fast_pow(a, (Q + 1) / 2, mod); //  $R = a^{((Q+1)/2)} \pmod{p}$ 
94
95 // 4. 主循环
96 while (t != 1)
97 {
98     if (t == 0) return 0; // a 是 0 的情况
99
100     // 找到最小的 i > 0 使得  $t^{(2^i)} = 1 \pmod{p}$ 
101     i64 i = 0;
102     i64 temp_t = t;
103     while (temp_t != 1)
104     {
105         temp_t = (i128)temp_t * temp_t % mod;

```

```
106         i++;
107     }
108
109     // 理论上不会发生, 除非输入p不是素数
110     if (i >= M) return -1;
111
112     // 计算 b = c^(2^(M-i-1))
113     i64 b_exp = 1LL << (M - i - 1); // 2^(M-i-1)
114     i64 b = fast_pow(c, b_exp, mod);
115
116     // 更新 M, c, t, R
117     M = i;
118     c = (i128)b * b % mod;
119     t = (i128)t * c % mod;
120     R = (i128)R * b % mod;
121 }
122
123 return R;
124 }
125 // snippet-end
126
127 void solve()
128 {
129 }
130 }
131
132 signed main()
133 {
134     // ios::sync_with_stdio(false);
135     // cout.tie(nullptr);
136     // cin.tie(nullptr);
137     int T = 1;
138     // cin >> T;
139     while (T--)
140         solve();
141     return 0;
142 }
```

5 附录

5.1 VSCode 设置

VSCode 常用设置

- Auto Save (自动保存)
- Alt 键的快捷键
- Code Runner 运行快捷键

- Run in Terminal
- Run in Terminal

- 有时间可选: smooth

5.2 互质的规律

互质规律: 比较常见的定义 1. 较大数是质数, 两个数互质

2. 较小数是质数, 较大数不是它的倍数, 两个数互质

3. 1 与其他数互质

4. 2 与奇数互质

一些推论 1. 两个相邻的自然数一定互质

2. 两个相邻的奇数一定互质

3. n 与 $2n + 1$ 或 $2n - 1$ 一定互质

求差判断法如果两个数相差不大, 可先求出它们的差, 再看差与其中较小数是否互质。如果互质, 则原来两个数一定是互质数。如: 194 和 201, 先求出它们的差, $201 - 194 = 7$, 因 7 和 194 互质, 则 194 和 201 是互质数。相反也成立, 对较大数也成立

求商判断法用大数除以小数, 如果除得的余数与其中较小数互质, 则原来两个数是互质数。如: 317 和 52, $317 \div 52 = 6 \dots 5$, 因余数 5 与 52 互质, 则 317 和 52 是互质数。

5.3 常数表

n	$\log_{10} n$	$n!$	$C(n, n/2)$	$\text{LCM}(1 \dots n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1.00000000	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
P_n	37338 ₄₀	204226 ₅₀	966467 ₆₀	190569292 ₁₀₀	1e9 ₁₁₄

$\max \omega(n)$: 小于等于 n 中的数最大质因数个数

$\max d(n)$: 小于等于 n 中的数最大因数个数

$\pi(n)$: 小于等于 n 中的数最大互质数个数

$n \leq$	10	100	1e3	1e4	1e5	1e6
$\max \omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \leq$	1e7	1e8	1e9	1e10	1e11	1e12
$\max \omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n \leq$	1e13	1e14	1e15	1e16	1e17	1e18
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x) \sim \frac{x}{\log(x)}$					

5.4 常见错因

爆数据 (爆 int, 爆 longlong)

取 mod 没有取干净或者取 mod 时超范围

想不出题事算一下各种数据范围

5.5 斐波那契数列

$$1. \sum_{i=1}^n F_i = F_{n+2} - 1.$$

$$2. \sum_{i=1}^n F_{2i-1} = F_{2n}.$$

$$3. \sum_{i=1}^n F_{2i} = F_{2n+1} - 1.$$

$$4. \sum_{i=1}^n F_i^2 = F_n F_{n+1}.$$

$$5. F_{n+m} = F_{n+1}F_m + F_nF_{m-1}.$$

$$6. F_{n-1}F_{n+1} - F_n^2 = (-1)^n.$$

$$7. F_{2n-1} = F_n^2 + F_{n-1}^2.$$

$$8. F_n = \frac{F_{n+2} + F_{n-2}}{3}.$$

$$9. F_{2n} = F_n(F_{n+1} + F_{n-1}).$$

$$10. \text{对任意 } k \in \mathbb{N}, \text{ 有 } F_n \mid F_{nk}.$$

(Cassini 恒等式)

$$11. \text{若 } F_a \mid F_b, \text{ 则 } a \mid b.$$

$$12. \gcd(F_n, F_m) = F_{\gcd(n, m)}.$$

$$13. F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right].$$

5.6 算法

出现的错误检查

1. 超出范围 (int, i64)
2. 取模出现错误
3. 图可能有重边或者自环

杂项 (通用)

想不到的题考虑:

1. 二分, 二分答案, 三分
2. dp
3. 离线
4. 倒推
5. 倍增
6. 建图

分块

一个二进制状态, 枚举他的所有子集状态 for(j = status; j > 0; j = (j - 1) & status);

中位数:

- 变成二分变成 +1 -1 判断
- 若动态维护, 每次加入一个数然后求中位数, 用对顶堆
- 如果还有删除一个数, 那么就用对顶 multiset

DP

- 背包
- 区间 dp
- DAG 图上的 dp
- 树形 dp
 - 直接 dfs
 - DFN 序:

- * 能够知道这个节点的子树大小
- * 某节点在不在此子树
- * 适用于对于一个节点向一个还没有合并过的节点合并复杂度较低的问题

- 换根 dp
- 基环树

- 状压 dp
- 数位 dp

搜索

- dfs, bfs
- bfs 不适用于有权图 01bfs 可适用于权值只有 0 和 1 的图
- 双向广搜: 两边各搜一半再合并

图论

- 拓扑排序:
 - 能够处理环, 算出环的大小, 链的大小等
- tarjan 缩点 (连通性相关)
 - 边双联通分量: 等价于: 该子图中任意两点之间至少存在两条边互不相交的路径
 - 点双联通分量: 等价于: 任意两点之间至少存在两条内部点互不相交的路径。

树

- 倍增
 - lca
- 树的重心
 - 定义:
 1. 以某个节点为根时, 最大子树的节点数最少, 那么这个节点是重心
 2. 以某个节点为根时, 每颗子树的节点数不超过总节点数的一半, 那么这个节点是重心
 3. 以某个节点为根时, 所有节点都走向该节点的总边数最少, 那么这个节点是重心
 - 性质:
 1. 一棵树最多有两个重心, 如果有两个重心, 那么两个重心一定相邻
 2. 如果树上增加或者删除一个叶节点, 转移后的重心最多移动一条边
 3. 如果把两棵树连起来, 那么新树的重心一定在原来两棵树重心的路径上

4. 树上的边权如果都为正数, 不管边权怎么分布, 所有节点都走向重心的总距离和最小

• 树的直径

- 求法:
 1. 两次 dfs 找两个距离最远的点不适用于有负边的树
 2. 树形 dp 对于每个点找子树中的最长的两条链适用于所有树

- 性质:

如果树上的边权都为正, 则有如下直径相关的结论:

1. 如果有多条直径, 那么这些直径一定拥有共同的中间部分, 可能是一个公共点或一段公共路径
2. 树上任意一点, 相隔最远的点的集合, 直径的两端点至少有一个在其中

• 树上差分

- 点差分
- 边差分

- 换根 dp
- 重链剖分
- 树上启发式合并
 - 适用于对多个子树统计答案
 - 树上启发式合并的特征:
 1. 没有修改操作
 2. 可以通过遍历子树, 建立信息统计, 得到所有查询的答案

数学

- 数论分块
- 互质的情况
- 素数密度
- 质数判断:
 1. 一个较小质数判断, 试除法
 2. 一个较大质数判断, miller rabin
 3. 一个范围内质数 (较多质数) 判断欧拉筛
- 质因数分解:
 1. 数量少用试除法 $O(\sqrt{n})$
 2. 数量多欧拉筛除以 minp

字符串

- kmp
- 字符串哈希
- trie

数据结构

- 链表, 栈, 队列
- **单调栈**: 能够知道每个数前面距离最近的比它大或者小的数
- **单调队列**: 能够知道每个长度为 k 的子区间的最值
- **堆**
 - 对顶堆: 能够方便的动态维护集合中第 k 大的元素
- **并查集**:
 - 扩展域/种类并查集: 能够维护满足 1. 朋友的朋友是朋友 2. 敌人的敌人是朋友
 - 带权并查集: 维护到根的距离并且取模能够达到类似扩展域并查集的效果
- **线段树**:
 - 区间加减
 - 区间修改 Tag 设一个 ip 变量代表是否修改
 - 势能分析直接暴力修改到叶子
 - 区间合并
 - 扫描线的修改, 区间懒标记是否被选取, 因为删除的时候只会删除已经添加过的区间
 - 动态开点
- **树状数组**
 - 能够维护可差分信息
 - 能够更快的边维护边查单个点的前缀
 - 能够动态地知道每个数前面有多少个小于它的数
 - kth 知道第 k 大的数是多少
 - 树上二分
- **波纹疾走树**
 - 查询区间第 k 小的数字
 - 区间内一个数字/一段数字的频率

5.7 组合数学公式

性质 1:

$$C_n^m = C_n^{n-m}$$

性质 2:

$$C_{n+m+1}^m = \sum_{i=0}^m C_{n+i}^i$$

性质 3:

$$C_n^m \cdot C_m^r = C_n^r \cdot C_{n-r}^{m-r}$$

性质 4 (二项式定理):

$$\sum_{i=0}^n (C_n^i \cdot x^i) = (1+x)^n$$

$$\sum_{i=0}^n C_n^i = 2^n$$

性质 5:

$$\sum_{i=0}^n ((-1)^i \cdot C_n^i) = 0$$

性质 6:

$$C_n^0 + C_n^2 + \dots = C_n^1 + C_n^3 + \dots = 2^{n-1}$$

性质 7:

$$C_{n+m}^r = \sum_{i=0}^{\min(n,m,r)} (C_n^i \cdot C_m^{r-i})$$

$$C_{n+m}^n = C_{n+m}^m = \sum_{i=0}^{\min(n,m)} (C_n^i \cdot C_m^i), \quad (r = n \mid r = m)$$

性质 8:

$$m \cdot C_n^m = n \cdot C_{n-1}^{m-1}$$

性质 9:

$$\sum_{i=0}^n (C_n^i \cdot i^2) = n(n+1) \cdot 2^{n-2}$$

性质 10:

$$\sum_{i=0}^n (C_n^i)^2 = C_{2n}^n$$

5.8 编译参数

-D_GLIBCXX_DEBUG : STL debugmode

-fsanitize=address : 内存错误检查

-fsanitize=undefined :UB 检查

5.9 运行脚本

Linux 运行脚本

```
#!/bin/bash  
g++ -std=c++20 -O2 -Wall "$1.cpp" -o "$1" -D_GLIBCXX_DEBUG  
./"$1" < in.txt > out.txt  
cat out.txt
```

5.10 随机素数

979345007 986854057502126921

935359631 949054338673679153

931936021 989518940305146613

984974633 972090414870546877

984858209 956380060632801307